Transformation eines technischen Systemmodells auf der Basis der Systemtheorie der Technik nach Ropohl in eine Simulations- und Modellierungs-Umgebung

Vom Fach Ingenieurwissenschaften
der Universität Duisburg-Essen
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)
genehmigte Dissertation.

von
Carsten J. Rudolph
aus Krefeld

Referent: Prof. Dr. rer. nat. W. Haupt

Korreferent: Prof. Dr.-Ing. E. Sauer

Tag der mündlichen Prüfung: 8. 8. 2006

Inhaltsverzeichnis

1 Kurzfassung

In dieser Arbeit wird ein Konzept vorgestellt, das die technische Handlungskompetenz von Schülern und Studenten nachhaltig fördern soll. Mit Methoden des systematischen Transformierens eines abstrakten in ein konkretes Systemmodell auf der Basis einer Simulations- und Modellierungs-Umgebung (SiMo-Umgebung) lernen die Schüler oder Studenten das gezielte Analysieren von technischen Systemen und deren Funktionalitäten kennen. Im Anschluss haben sie die Möglichkeit, weitere Kenntnisse über das Verhalten des selbst erstellten Systems zu gewinnen und das System zu modifizieren und zu erweitern.

Das im Folgenden beschriebene Konzept basiert auf der Systemtheorie der Technik nach Ropohl [Ropohl, 1999]. Das zu betrachtende technische System wird in mehreren Schritten von der abstrakten Darstellung in ein virtuelles technisches System transformiert, das simuliert und modelliert werden kann.

Der erste Schritt ist die Erstellung eines abstrakten Systemmodells nach Ropohl ausgehend von einem realen technischen System.

Dieses Modell wird in einem zweiten Schritt isomorph in ein Modell auf der Basis von UML (Unified Modeling Language) transformiert. Diese Transformation wird durch Benennung von Funktionen konkretisiert.

Zu Beginn des dritten Schrittes wird das bereits erweiterte Modell isomorph in eine objektorientierte Programmiersprache[1] transformiert. Anschließend wird es soweit konkretisiert und modifiziert, dass es zu einem virtuellem technischen System wird, das simuliert und modelliert werden kann.

1 In dieser Arbeit wird die objektorientierte Programmiersprache **Java** der Firma **Sun microsystems®** verwendet.

2 Einleitung

Das in dieser Arbeit entwickelte Konzept zur Förderung technischer Handlungskompetenz entstand während des Projektes „*Entwicklung von Lehr- und Lernmodulen im Baukastenmodus zu den disziplinübergreifenden Bereichen 'Stoff-, Energie- und Informationsumsetzende Systeme' für die Studiengänge Lehrämter Technik Sekundarstufe I und Sekundarstufe II*“ im Rahmen des Förderprogramms „*Neue Medien in der Hochschullehre*“, gefördert durch das **BMBF** (Bundesministerium für Bildung und Forschung) [BMBF].
Ziel des Projektes war es, Lehrern und Lernern[2] Lernobjekte zur Verfügung zu stellen, mit denen sie ihre Lehr- und Lerneinheiten individuell gestalten können. Dieses Ziel ergab sich aus den jahrelangen Erfahrungen mit der Gestaltung von Lernszenarien im Fach **TUD** an der Universität Duisburg-Essen, Campus Essen.
Das Fach **TUD - Technologie und Didaktik der Technik** an der Universität Duisburg-Essen, Campus Essen, im Fachbereich 12 Maschinenwesen hat in erster Linie die Aufgabe, Studenten auf das Erste Staatsexamen für das Lehramt Technik an Grund-, Haupt- und Realschulen und den entsprechenden Jahrgangsstufen der Gesamtschulen und das Lehramt an Gymnasien und Gesamtschulen[3] vorzubereiten. Die wesentliche Intention besteht darin, den zukünftigen Techniklehrern eine allgemeine technische Bildung zu vermitteln. Ein Teil dieser Bildung besteht im Erlernen technischen Denkens und Handelns. Dazu gehört unter anderem, technische Systeme zu analysieren und zu bewerten.

Zu Beginn des Projektes wurden Kriterien herausgearbeitet, die diese Lernobjekte erfüllen sollten, damit sie definierte Ansprüche erfüllen konnten (vergl. [Wehling, Langkau, Rudolph, Haupt, 2003], [Langkau, Rudolph, Wehling, 2003], [Wehling, 2003], [Langkau, Rudolph, Wehling, Haupt, 2002], [TUD-Ein-

2 Die Begriffe „Lehrer“ und „Lerner“ werden hier in ihrer allgemeinen Bedeutung verwendet.

3 Die neuen Lehrämter gelten seit dem Wintersemester 2003/2004. Sie ersetzen die alten Lehrämter für die Primarstufe und die Sekundarstufen I und II.

gangsportal-2]). Da abzusehen war, dass bei dieser Vorgehensweise eine Fülle von Objekten entstehen würde, war die Notwendigkeit einer geeigneten Archivierung und Distribution dieser Objekte gegeben. Dieses Problem wurde mit

einer eigens für und im Fach **TUD** entwickelten **MultiMediaDatenBank-Technikunterricht** (**MMDB-TU**) gelöst (vergl. [TUD-Eingangsportal-3], [Wehling, 2003]).

Der Einsatz dieser digitalen Lernobjekte erforderte im Weiteren, dass für das Fach **TUD** eine Lernplattform eingerichtet werden musste, um diese Lernobjekte in den individuellen Seminaren allen Teilnehmern bereitstellen zu können (vergl. [TUD-Eingangsportal-3]).

Die Erfahrungen mit digitalen Lernobjekten zeigten, dass hauptsächlich aufwändige Lernobjekte, wie z. B. Animationen und Simulationen einen Mehrwert beim Lernen erbrachten (vergl. [Issing, 1997], [Issing, 1998]). Da die Beschaffung dieser Lernobjekte sehr kostenintensiv ist (vergl. [Wisser, 2002]), entschied das Fach **TUD**, diese mit eigenen Mitteln selbst zu erstellen. Es zeigte sich schnell, dass für die Erstellung derartiger Objekte ein sehr großes Potenzial an Erfahrung und Kenntnissen über die zu erstellenden Objekte erforderlich war. Daraus ergab sich die Idee, die Lernprozesse, die sich im Verlauf der Auseinandersetzung mit den zu erstellenden Objekten zwangsläufig ergeben, zu nutzen, um Lerner bei der Entwicklung technischer Handlungskompetenz zu unterstützen. Dieser Ansatz wird durch Untersuchungsergebnisse bestätigt (vergl. [Bresges, 2002]).

Diese Lernprozesse sollten jedoch durch ein Konzept begleitet werden, damit sie zielsicher und effektiv ablaufen können.
Die Zielsetzung des hier beschriebenen Konzeptes ist, Studenten mit Hilfe einer systematischen Systemanalyse bis hin zur Programmierung von technischen Systemen beim Erwerb einer fundierten technischen Handlungskompetenz zu unterstützen. Dieses Konzept ist sowohl für Schüler allgemein bildender Schulen als auch für Studenten geeignet.

In diesem Konzept wird die Systemtheorie der Technik nach Ropohl (vergl. [Ropohl, 1999]) als Basis für die Systemanalyse und spätere Modellbildung des zu betrachtenden Systems verwendet.

Auf der Basis eines abstrakten Systemmodells wird der Lerner über die Systemanalyse, die konkretere Beschreibung des Systems mittels **UML** und weiter über die vollständige Konkretisierung mit Hilfe der Programmierung in der objektorientierten Programmiersprache Java geführt.

Auf diese Weise wird der Lerner unterstützt, Wissen ziel- und problemorientiert, selbst organisiert, fallbezogen und eigenverantwortlich anzueignen. Es wird vermieden, dass er sich mit „*trägem Wissen*“ belastet [Mandel, 2001].

Bresges verfolgte in seiner Arbeit ähnliche Ziele und teilweise ähnliche Lösungsansätze. Sein Hauptziel war, multimediale Modelle technischer und naturwissenschaftlicher Systeme zu produzieren. Für diesen Zweck suchte er nach einer universell einsetzbaren Modellierungs-Methode. Aus diesem Grunde schieden spezielle Modellansätze von vornherein aus, unter anderem auch das Systemmodell nach Ropohl. Bresges verwendete bei seiner der Modellierung ausschließlich UML. Dies setzte bereits seitens der Lerner die Fähigkeit des Modellierens voraus.
Im Rahmen des hier vorgestellten Konzeptes ist jedoch gerade der Erwerb dieser Fähigkeit des Modellierens ein grundlegender Bestandteil.

3 Technische Handlungskompetenz

In den nächsten Abschnitten wird erläutert, welches Wissen und welche Fähigkeiten gefördert werden sollen, um die technische Handlungskompetenz der Lerner erweitern zu können.

3.1 Technisches Wissen

In dieser Arbeit wird der Begriff „*technisches Wissen*“ nach Ropohl [Ropohl, 1999] zugrunde gelegt. Ropohl gliedert technisches Wissen in fünf Unterkategorien. Sie lauten *technisches Können*, f*unktionales Regelwissen*, *strukturales Regelwissen*, *technologisches Gesetzeswissen* und *öko-sozio-technologisches Systemwissen (s. Abb. 1)*

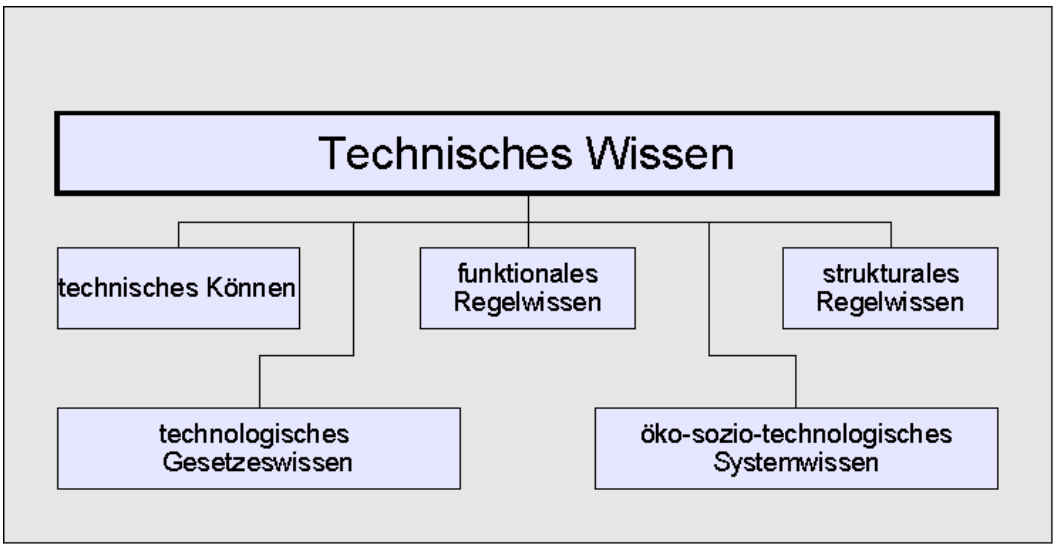


Abb. 1 Technisches Wissen nach Ropohl
Quelle: [Rohpol 1999, S. 201]

Unter **technischem Können** werden sowohl die Fertigkeiten und Geschicklichkeiten des Menschen verstanden, die charakterisieren wie er mit technischen Systemen umgeht (Bedienung), als auch die Fertigkeiten, wie er technische Systeme herstellt. Dabei sind die kognitiven Anteile der Handlungskompetenz sehr eng mit den motorischen Fähigkeiten verknüpft. Das technische Können beruht zum größten Teil auf den Erfahrungen des Individuums.

Das **funktionale Regelwissen** betrachtet das System aus der funktionalen Perspektive (s. Kapitel 4. Das Sachsystem wird als „*black box*“ betrachtet. Dieses Wissen lässt die Ursachen für das Verhalten des Systems außer Acht und betrachtet hauptsächlich seine Wirkung. Auch dieses Wissen wird hauptsächlich aus Erfahrung gewonnen.

Das **strukturale Regelwissen** hingegen betrachtet das System aus der strukturalen und aus der hierarchischen Perspektive (s. Kapitel 4). Dabei wird der innere Aufbau der Systeme betrachtet. Die Abhängigkeiten und Strukturen der einzelnen Subsysteme werden berücksichtigt. Dieses Wissen beruht wiederum auf Erfahrungen des Individuums, jedoch wird dieses Wissen dem Individuum durch andere übermittelt.

Die bis hier genannten Wissensniveaus reichen in der Regel aus, um bestehende Systeme zu bedienen und zu warten, jedoch erlauben sie noch keine konkrete Analyse eines betrachteten Systems oder die Entwicklung eines neuen Systems. Das System kann mit diesem Wissen nur abstrakt beschrieben werden.
Um es vollständig zu verstehen, ist das **technologische Gesetzeswissen** notwendig, das mit dem *naturwissenschaftlichen Wissen* annähernd vergleichbar ist. Das *technische Wissen* unterscheidet sich nur in einigen Punkten von dem *naturwissenschaftlichen Wissen*, z. B. teilweise in den Methoden der Erkenntnissgewinnung oder in der Zielsetzung (vergl. [Ropohl, 1981]).
Das technologische Gesetzeswissen umfasst sowohl die funktionalen und strukturalen Zusammenhänge des Systems als auch die naturalen Effekte, auf denen die Funktions- und Strukturprinzipien des Systems basieren. Mit diesem Wissen werden hauptsächlich von Ingenieuren komplexe Problemstellungen gelöst, z. B. Systemanalysen, Konstruktion neuer Systeme, Optimierung von Systemen, etc.

Im Fall von **öko-sozio-technologischem Systemwissen** wird das System unter ökologischen und soziologischen Aspekten betrachtet. Hier werden Erkenntnisse über das Verhalten und die Auswirkungen eines technischen Systems in der Natur und der Gesellschaft gewonnen. Dieses Wissen beruht zwar auch

hauptsächlich auf Erfahrungen, diese werden jedoch im Nachhinein mit Hilfe von mathematischen Modellen zu beschreiben versucht, wie z. B. Prognosen zum Treibhauseffekt.

Die Fähigkeiten, die mit der SiMo-Umgebung erworben werden können, lassen sich dem **technologischen Gesetzeswissen** zuordnen. Es handelt sich dabei um die Fähigkeiten des *Simulierens* von bestehenden Systemen und des *Modellierens* von neuen Systemen. Bei diesen Fähigkeiten ist das Wissen über die hierarchischen, strukturalen und funktionalen Zusammenhänge des Systems von großer Bedeutung und muss mit dem Wissen über die naturalen Gesetzmäßigkeiten in Form von Formeln verknüpft werden, um sinnvolle Ergebnisse zu erzielen.

3.2 Die technische Fähigkeit des *Simulierens*

„Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“ [VDI-Richtlinie 3633].

Unter **Simulieren** soll hier verstanden werden, dass ein technisches System gegeben ist, bei dem nur die Zustände (Eigenschaften) über die Input-Parameter beeinflusst werden können. Die Werte der Zustands- und Outputgrößen werden in beliebiger Form dargestellt. Die Realitätsnähe des simulierten Systems hängt entscheidend von den berücksichtigten Input-Größen und dem zugrunde liegenden Rechenmodell ab.

Beim Simulieren von technischen Systemen soll der User beurteilen, ob das System so arbeitet, wie es in der Zielvorgabe verlangt wird. Falls nicht, muss er die richtigen Parameter einstellen können, damit das System entsprechend den Zielvorgaben arbeiten kann. Um hier gezielt vorgehen zu können, muss ein grundlegendes Systemverständnis seitens des Users vorhanden sein.

Mit der Methode des Simulierens steht dem Lerner eine Lernmethode zur Verfügung, die ihm erlaubt, selbständig sein Wissen und das Verständnis für das System zu überprüfen. Das Verstehen zeigt sich, wenn im Falle von Parameteränderungen seine Vermutungen über die Verhaltensänderungen des Systems mit denen der Simulation übereinstimmen.

3.3 Die technische Fähigkeit des *Modellierens*

Bevor ein technisches System simuliert werden kann, muss es erst modelliert werden. Bei der Modellierung erstellt der User aus vorgegebenen Subsystemen ein Systemmodell, das im Anschluss simuliert werden soll. Die Komplexität und Funktionalität des Gesamtsystems hängen von der Definition der einzelnen Subsysteme und deren Möglichkeiten zur Interaktion mit den anderen Subsystemen ab. In Bezug auf die Realitätsnähe des Systems gelten weiterhin dieselben Anmerkungen wie im Falle der Simulation.

Beim Modellieren bekommt der User tiefer greifende Möglichkeiten das System zu verändern. Bei der Simulation kann der User nur auf der Ebene des Systems agieren; bei der Modellierung hat er noch zusätzlich die Option, Subsysteme anzuordnen und interagieren zu lassen. Dies stellt somit einen gesteigerten Anspruch an die technische Handlungsfähigkeit dar, da die Handlungen selbst komplexer geworden sind.

3.4 Die technische Fähigkeit des *Programmierens*

Die Fähigkeit, technische Systeme mit Hilfe einer Programmiersprache abzubilden, erfordert vom Programmierer ein fundiertes Wissen über das System und dessen konkreten inneren Aufbau. Mit Hilfe der Programmierung kann sehr gut erfahren werden, ob das technische System bezüglich seiner Funktionalität und seines inneren Aufbaus verstanden worden ist. Dabei muss sich der Programmierer immer wieder die Frage stellen, ob die Ergebnisse seiner erstellten Simulation in Vergleich zu den Messergebnissen eines realen Systems in einem angemessenen Fehlertoleranzbereich bleiben oder ob das Modell verfeinert werden muss.

4 Die *Systemtheorie der Technik* nach Ropohl

Die **Allgemeine Systemtheorie** ist eine interdiziplinäre Theorie, die sich für jeweilige Fachdiziplinen spezialisieren lässt (z. B. in der Soziologie, Biologie, Technik, usw.). Das Denken in Systemen reicht bis Aristoteles zurück und wurde im Laufe der Zeit immer weiter entwickelt. Die moderne Systemtheorie hat sich aus vier Wurzeln entwickelt (vgl. [Ropohl, 1999]).

Die erste Wurzel ist die **Allgemeine Systemlehre** des Biologen *Ludwig von Bertalanffy* in den 1930er Jahren. Er bemühte sich um eine Synthese zwischen 'mechanistischen' und 'vitalistischen' Vorstellungen und fand diese im Systemkonzept.
„Die Eigenschaften und Verhaltensweisen höherer Ebenen sind nicht durch die Summation der Eigenschaften und Verhaltensweisen ihrer Bestandteile erklärbar, solange man diese isoliert betrachtet. Wenn wir jedoch das Ensemble der Bestandteile und Relationen kennen, die zwischen ihnen bestehen, dann sind die höheren Ebenen von den Bestandteilen ableitbar“ [Bertalanffy, 1949].

Die **Kybernetik**, wie sie 1948 von N. Wiener vorgeschlagen wurde, kann als zweite Wurzel bezeichnet werden. Obwohl in der Kybernetik nie ausdrücklich auf die Systemtheorie verwiesen wurde, war das Systemdenken von Anfang an prinzipiell eingeschlossen [Ropohl, 1999].

Die dritte Wurzel ist in der **Verwissenschaftlichung praktischen Problemlösens** zu sehen. Reale komplexe Probleme lassen sich selten mit **einer** wissenschaftlichen Theorie lösen, sondern erfordern die Erkenntnisse aus weiteren Teildiziplinen. Da somit die Teilaspekte und ihre Zusammenhänge der unterschiedlichen Fachdiziplinen erforderlich werden, kann das Problem nur in seiner Ganzheit betrachtet gelöst werden, und es muss somit ein System aus wissenschaftlichen Aussagen gebildet werden, die der Komplexität des realen Problems entsprechen [Ropohl, 1999].

Erste Vorläufer derartiger Bemühungen sind das „*Operations Research*“, eine Form der mathematischen Modellanalyse zur Optimierung praktischer Lösungen und das „*System Engineering*“, das als Erstes in Amerika zur Analyse und Planung großer Telefonnetze eingesetzt wurde [Ropohl, 1999].

Die vierte Wurzel lässt sich schließlich im **strukturalen Denken der modernen Mathematik** finden. Mathematik versteht sich heute als Wissenschaft der allgemeinen Strukturen und Relationen, teilweise sogar als Strukturwissenschaft, somit wird sie nicht nur zum Werkzeug der Systemtheorie, sondern vielmehr zur Systemtheorie überhaupt [Ropohl, 1999].

Ropohl versteht seine *Systemtheorie der Technik* als Beschreibungsinstrument, mit dem es gelingt, die verschiedenen Perspektiven der Technik in Relationen zu setzen. Er betrachtet die Systemtheorie als eine Modelltheorie, mit der die verschiedenen Wirklichkeitsbereiche in derselben Sprache beschrieben werden und dadurch aufeinander bezogen werden können. Dabei unterscheidet er zwischen drei Aspekten, die jeweils eine unterschiedliche Perspektive auf das System darstellen, und zwar den hierarchischen, den strukturalen und den funktionalen Aspekt [Ropohl, 1999].

4.1 Der hierarchische Aspekt

Der hierarchische Aspekt beschreibt die **Hierarchie** der einzelnen Systeme untereinander. Ein System, das in einem anderen System enthalten ist, wird Subsystem genannt. Ein System, das einem anderen übergeordnet ist, wird als Supersystem bezeichnet. Ob ein System als Super- oder Subsystem bezeichnet wird, hängt von dem Bezugssystem ab, von dem aus die Hierarchie betrachtet wird. Wird das Bezugssystem in Richtung der Subsysteme analysiert, so können detaillierte Informationen über das System erhalten werden. Geht die Analyse in Richtung der Supersysteme, so können umfassende Erkenntnisse über die Bedeutung der Bezugssysteme gewonnen werden. Mit diesem Konzept lassen sich sowohl tiefer gehende Erkenntnisse über das System gewinnen als auch eine immer weiter greifende Synthese von Zusammenhängen anwenden [Ropohl, 1999].

4.2 Der strukturale Aspekt

Der strukturale Aspekt betrachtet die **Relationen** der einzelnen Subsysteme untereinander. Er geht davon aus, dass etwas nur verstanden werden kann, wenn die anderen Systeme, die mit dem betrachteten System in Relation stehen, mit in die Betrachtungen einbezogen werden [Ropohl, 1999].

4.3 Der funktionale Aspekt

Der funktionale Aspekt bezieht sich auf die **Funktionalität** des Systems und ist im Allgemeinen jedem Techniker bekannt. Es betrachtet das System als „*black box*"' und analysiert dessen Inputs und Outputs. Über die Inputs und Outputs können die Eigenschaften (auch Zustände genannt) des Systems oder anderer Systeme manipuliert werden. Dieses basiert auf dem alltäglichen Umgang mit technischen Systemen: Der Anwender drückt auf einen Knopf und erwartet eine Leistung, die das System ihm erbringen soll. Das funktionale Systemdenken dringt nicht in die materielle Konkretisierung und den inneren Aufbau des Systems ein. Es beschränkt sich auf das Verhalten des Systems in seiner Ganzheit innerhalb seiner Umgebung. Das System selbst wird nicht behandelt, sondern lediglich sein Verhalten [Ropohl, 1999].

4.4 Zusammenfassung

Fasst man diese drei Aspekte zusammen, so lässt sich nach Ropohl ein System so definieren:

„Ein System ist das Modell einer Ganzheit, die (a) Beziehungen zwischen Attributen (Inputs, Outputs, Zustände etc.) aufweist, die (b) aus miteinander verknüpften Teilen bzw. Subsystemen besteht, und die (c) von ihrer Umgebung bzw. von einem Supersystem abgegrenzt wird.

Mit dieser Definition sind die drei Systemaspekte [...] miteinander vereinigt; (a) definiert die Funktionen, (b) die Struktur und (c) die Hierarchie eines Systems.“ ([Ropohl, 1999].

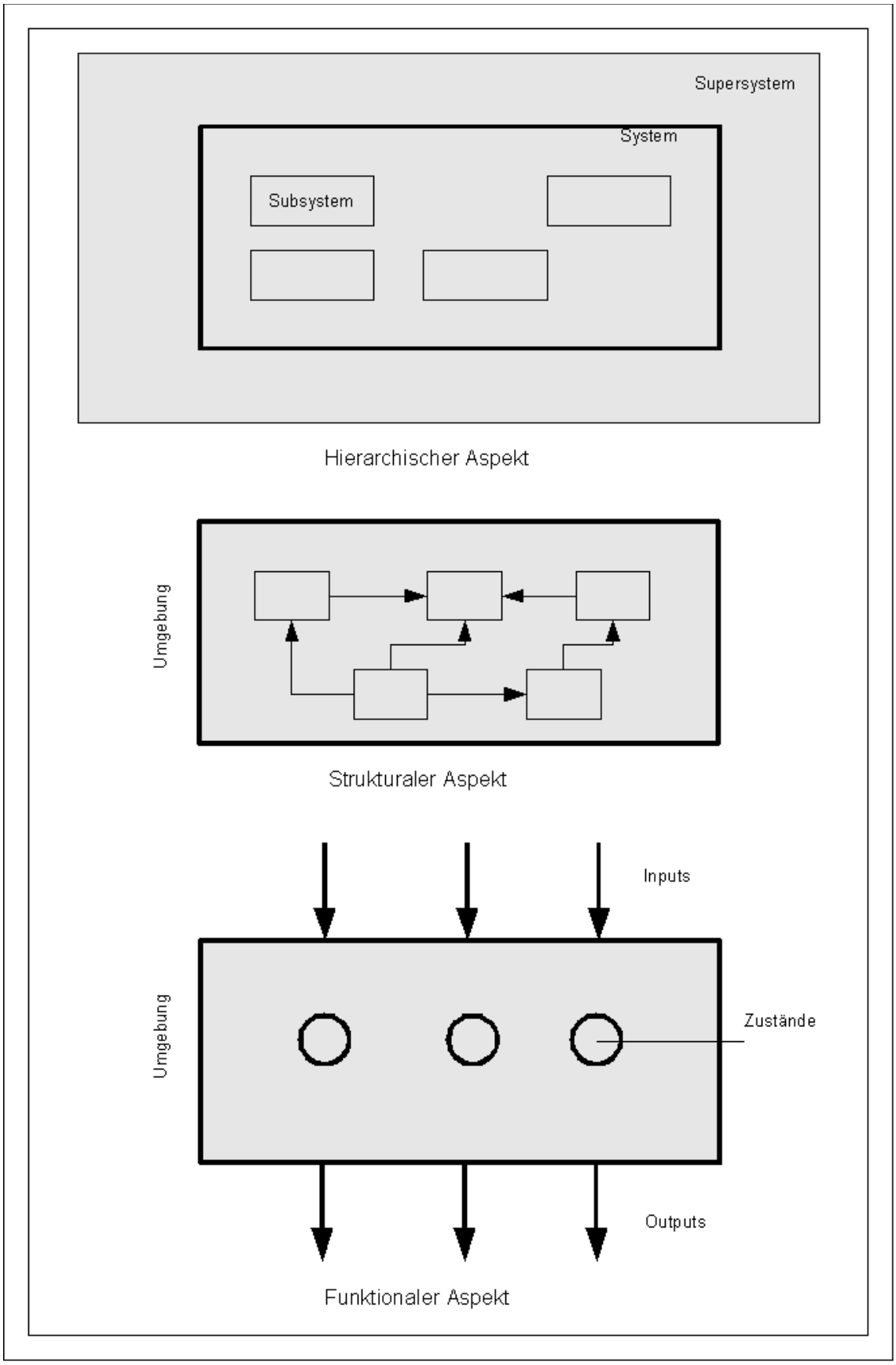


Abb. 2 Aspekte eines technischen Systems

4.5 Vorzüge der Systemtheorie der Technik zur Modellbildung

Es gibt zwei nahe liegende Gründe das Systemkonzept nach Ropohl als Basis für das hier vorgestellte Konzept auszuwählen.

1. Das Systemkonzept nach Ropohl ist Bestandteil der Lerninhalte des Faches TUD. Mit seiner Hilfe werden technische Systeme analysiert. Durch Verwendung des in dieser Arbeit vorgestellten Konzeptes kann der praktische Anwendungsbereich des Ropohlschen Systemkonzeptes stark erweitert werden.

2. Die Darstellung eines technischen Systems mit Hilfe der Modellsymbolik ist aufgrund von der geringen Anzahl von Symbolen recht einfach. Dennoch lassen sich durch hierarchische, strukturale und funktionale Anordnungen von Subsystemen komplexe Systemstrukturen abbilden.
   Um die Einfachheit zu erhalten, fielen Erweiterungen dieser Modellbildung aus der Auswahl heraus (z. B. [Arp, 1991]; [Arp, 2000]). Die erweiterten Modelle zielen auf eine komprimiertere und komplexere Darstellung von Systemzusammenhängen und bedienen sich dabei einer erweiterten Symbolik.
   Andere Modellkonzepte, wie z. B. Modellkonzepte der Thermodynamik oder der Elektrotechnik, wurden nicht ausgewählt, da diese Konzepte speziell für diese Fachgebiete ausgelegt sind.

Alle Systemmodellkonzepte haben eine gemeinsame Schwäche. Sie können entweder die Strukturen eines Systems abbilden, dabei aber die zeitlichen Änderungen im System nicht berücksichtigen, oder sie erfassen die zeitlichen Änderungen bestimmter Systemgrößen, verzichten jedoch auf die Darstellung der Systemstrukturen (vergl. [Bresges, 2002]).

Ein System kann mit Hilfe weniger Symbole unter drei verschiedenen Aspekten (hierarchischer, strukturaler und funktionaler Aspekt) quantitativ vollständig beschrieben werden. Durch systematische Konkretisierungen der einzelnen Subsysteme kann das System auf einfache Weise bis in das kleinste Subsystem zerlegt werden. Der Vorteil bei dieser Vorgehensweise ist, dass zu jedem Zeit-

punkt ein vollständiges Modell des zu betrachtenden Systems gegeben ist. Der Unterschied liegt in der Komplexität der Darstellung des gesamten Systems, diese kann vom Betrachter selbst bestimmt werden.

5 Objektorientierte Programmiersprache

Programmiersprachen wie **C++**, **Java**, **VisualBasic**, **VisualC** und die meisten anderen neueren Programmiersprachen und Skriptsprachen versuchen, sich an der **objektorientierten Programmierung** (**OOP**) zu orientieren. **Java** erhebt dabei den Anspruch, eine „richtige“ objektorientierte Programmiersprache zu sein. Sie ist dem C++ sehr ähnlich, aber in Detailfragen wurden einige Architekturen und Funktionen von C++ anders gelöst mit dem Ziel, einfacher und weniger fehleranfällig zu sein als C++. Für diese Arbeit ist von besonderem Interesse, dass im Bereich der Objekte-Verwaltung und im Umgang mit Threads vieles durch vorgegebene Funktionen vereinfacht und stabiler gestaltet wurde als in C++. Zusammenfassend kann gesagt werden, dass Java leichter zu erlernen ist als C++, ohne dass dadurch zu große Einschränkungen im Einsatzbereich der Sprache hinzunehmen sind.
Das bekannteste und wichtigste Argument für die Verwendung von Java ist, dass es auf fast allen bekannten Plattformen (Betriebssystemen), auf denen die **Java Runtime Environment** (JRE) installiert ist, lauffähig ist.

5.1 OOP - Objektorientierte Programmierung

Der objektorientierte Ansatz stammt aus den 1970er Jahren (1970-1980: PARC **-** Palo Alto Research Center der Firma XEROX). Die Ursprünge der OOP wurden von der Firma XEROX in der Sprache Smalltalk-80 entwickelt. Später wurde in der Programmiersprache SIMULA 67 das Klassenkonzept übernommen und weiter entwickelt. Erst nach zehnjähriger Erfahrung mit OOP kam der Durchbruch für die objektorientierte Denkweise bei der Programmierung von Anwendungen (vergl. [Balzert, Helmut, 2001(1)]).

OOP setzt eine neue Denkweise voraus. Diese Denkweise unterscheidet sich wesentlich von den herkömmlichen Denkweisen bei der Problemlösung. Sie geht von Objekten aus, die miteinander zu jeder Zeit in Interaktion treten, während die herkömmliche Denkweise von zwar verzweigten aber dennoch zeitlich linearen Programmabläufen ausgeht. Diese objektorientierte Denkweise lässt sich deshalb sehr gut mit den Grundüberlegungen der im Kapitel 4 be-

schriebenen Systemtheorie kombinieren. Es lassen sich in der OOP die Betrachtungsweisen von Systemen wiederfinden, wie sie in der Systemtheorie verwendet werden. Zur besseren Unterstützung der Planung und Programmierung von objektorientierten Anwendungen wurde die **Unified Modeling Language (UML)** entwickelt. Sie erlaubt es, die Denkweise der **OOP** in graphischen Elementen wie z. B. Diagrammen zu visualisieren.

Im Mittelpunkt der **OOP** steht das **Objekt**. Ein Objekt besteht aus den drei Elementen **Name**, **Eigenschaften** und **Methoden**. Der **Name** gilt als eindeutiger Identifier für das Objekt. **Eigenschaften** beschreiben den Zustand des Objektes. Der Wert der jeweiligen Eigenschaft wird als **Attribut** bezeichnet. So ist z. B. '***Farbe***' eines Objektes die **Eigenschaft,** und '***grün***' das **Attribut**. Eigenschaften können selbst wiederum durch Objekte beschrieben werden. Es müssen nicht unbedingt einfache Datentypen sein.
Die **Methoden** eines Objektes beschreiben seine Funktionalitäten. Über diese werden die **Eigenschaften** des Objektes manipuliert. Die Methoden sind der einzige Zugang von außerhalb des Objektes, um auf die Eigenschaften zuzugreifen. Dies nennt man in der OOP „*Kapselung*“.

Ein **Objekt** wird über eine **Klasse** definiert. Die Objekte, die während der Laufzeit der Anwendung in Aktion treten, werden erst während der Laufzeit aus den Klassen gebildet.
Objekte können über ihre Eigenschaften und Methoden in Abhängigkeiten gesetzt werden.
Ein Vergleich zwischen einem Objekt und einem System[4] zeigt, dass ein Objekt ein System beschreiben kann. Da ein Objekt auch mehrere Objekte zusammenfassen kann, können auch Objekthierarchien gebildet werden. Die konkrete Umsetzung wird im weiteren Verlauf der Arbeit ausführlich beschrieben.

In diesem Abschnitt konnte gezeigt werden, dass das oben beschriebenen Systemkonzept sich problemlos in die OOP überführen lässt, da dieselben Strukturen vorhanden sind. Es müssen lediglich einige Begriffe ausgetauscht werden.

4 Systembegriff nach [Ropohl, 1999]

5.2 UML - Eine Sprache die verbindet

Unter diesem Abschnitt soll erläutert werden, was UML ist, inwieweit auf UML in dieser Arbeit zurückgegriffen wird, und wie das Konzept dieser Arbeit mit Hilfe von UML in Zukunft noch einfacher umsetzbar wird.

Die **Unified Modeling Language** (**UML**) wurde unter der Koordination des WWW-Consortiums entwickelt, um Anwendungen auf der Basis des objektorientierten Denkens besser erstellen zu können. Zu diesem Zweck wurden verschiedene Diagrammtypen entwickelt, die Objekte, ihre Relationen, Zustände und Aktivitäten in verschiedenen Diagrammformen darstellen können. Mit Hilfe dieser Diagramme kann aus verschiedenen Perspektiven auf das System der Objekte geblickt werden. Die Diagramme stehen dabei immer in bestimmten Abhängigkeiten zueinander. Erst durch die Gesamtheit all dieser Diagramme wird das System und die Problemlösung mit Hilfe dieses Systems vollständig und konkret beschrieben. Diese Darstellung wurde gewählt, damit alle Beteiligten (Kaufleute, Techniker, Ingenieure, Programmierer, usw.) die bei einer Problemlösung mitarbeiten, sich in einer gemeinsamen Sprache miteinander verständigen können. Das Revolutionäre an der UML ist, dass es durch die OOP möglich ist, anhand der Informationen aus den verschiedenen Diagrammen eine fertige Anwendung (Problemlösung) durch Softwaretools generieren zu lassen. Dabei kann sogar ausgewählt werden, in welcher gängigen, objektorientierten Programmiersprache die Ausgabe erfolgen soll.

Diese UML-gestützten Entwicklungsumgebungen haben schon einen ausgesprochen hohen Entwicklungsstand erreicht. Leider sind die Kosten der Entwicklungsumgebungen so hoch, dass sie zur Zeit für die Lehre noch unerschwinglich sind. Mit Hilfe dieser Entwicklungsumgebungen könnte dieses Konzept größtenteils ohne das Erlernen von Java auskommen. Das didaktische Konzept ließe sich noch einfacher umsetzen.

5.3 Vorzüge der OOP

Das Erlernen einer objektorientierten Programmiersprache bietet folgende Vorteile.

- Es können beliebige Systeme abgebildet werden.
- Es gibt keine Beschränkung in der Komplexität und der Erweiterbarkeit.
- Die Funktionalitäten können frei bestimmt werden. Es ist möglich, die selbst erstellten Systeme über Hardwareinterfaces mit realen Systemen zu koppeln.

Hinzu kommen weitere Argumente für die Programmiersprache Java:

- ständige Weiterentwicklung der Sprache durch die Community, gesponsert von **SUN microsystems®,**
- kostenfreie Distribution der Programmiersprache Java (freie Lizenzen),
- kostenfreie, mächtige Entwicklungsumgebungen zur Erstellung von Java-Applikationen (z. B. Netbeans [NetBeans], Eclipse [Eclipse]),
- Unterstützung bei der Planung und Generierung von Java-Programmcodes durch UML-Tools (sind zum jetzigen Zeitpunkt entweder zu teuer oder noch nicht ausgereift, aber die Entwicklung geht schnell und stetig voran),
- Weltweite Unterstützung der Entwicklung der UML-Tools durch zahlreiche Communities
- Java wird vielfach in der Industrie verwendet. Beispiel VW: Hier laufen Projekte, in denen versucht wird, komplette Produktionsabläufe in einer objektorientierten Sprache abzubilden, um Fehler im Produktionsablauf vorzeitig zu erkennen.
- Der Lerner erlernt die objektorientierte Denkweise, die sich mittlerweile nicht nur bei der Entwicklung technischer Systeme durchgesetzt hat. Produktions- und Verfahrensabläufe werden heute mit Hilfe der objektorientierten Denkweise geplant und durchgeführt.

Diese Punkte zeigen auf, dass das Erlernen von Java eine Investition in das „*lebenslange Lernen*“ ist. Java in Kombination mit dem entwickelten Konzept ermöglicht es, an technische Probleme strukturiert und zielsicher heranzugehen.

6 Abgrenzung der SiMo-Umgebung von bestehenden Simulations- und Modellierungsprogrammen

6.1 Simulations- und Modellierungsprogramme

In diesem Abschnitt soll zwischen zwei Kategorien von Simulationsprogrammen unterschieden werden. Einerseits den bekannten Simulationen, die ein bestimmtes System simulieren (z. B. Flugsimulator, Rennwagensimulatoren usw.) und andererseits den Systemmodellierungsprogrammen, mit denen in einer abstrakten Symbol- und Formelsprache Systeme modelliert und simuliert werden können (Dynasys: vergl. [DYNASYS]; Stella: vergl. [STELLA]). Die Stärken und Schwächen dieser beiden Kategorien liegen in ihren Anwendungszielen begründet. Soll die Simulation über ein System ausführliche und zuverlässige Auskünfte geben und diese auch im Sachzusammenhang entsprechend darstellen (beim Flugsimulator wird z. B. der Wahrnehmungsbereich eines Piloten im Cockpit dargestellt), so beschränkt sich die Darstellung bei den Systemmodellierungs-Tools größtenteils auf mathematische und grafische Darstellungen. Die Genauigkeit hängt sowohl von den Möglichkeiten des gesamten Systemmodellierungs-Tools als auch von den Ansprüchen des Users ab.
Die Berechnungsmethoden des ersten Simulationstyps sind speziell für den Zweck der Simulation optimiert. Das Systemmodell der Simulation ist für den in die Entwicklung der Simulation nicht involvierten User nicht erkennbar. Er kann nur die Reaktionen des Systems auf seine Aktionen erfassen.

Anders ist es bei den Systemmodellierungs-Tools. Hier kann der User über die Symbol- und Formelsprache des Tools eigene kleine Systemmodelle erstellen und simulieren. Die Werte der einzelnen Parameter kann er dann in vorbestimmten Darstellungsformen anzeigen lassen. Der Handlungsspielraum des Users wird durch die Möglichkeiten der bereitgestellten Tools beschränkt. Er ist auf die hinterlegten Berechnungsverfahren festgelegt und kann diese nicht verändern.
Diese Einschränkungen verhindern, dass Systeme, die mit einem Systemmodellierungs-Tool erstellt wurden, später durch weitere Konkretisierungen und

Erweiterungen (z. B. Verknüpfung mit realen Systemen über Hardwareinterfaces) erweitert und weiter entwickelt werden können. Sollt dies geschehen, wäre ein Toolwechsel nötig.

Zusammenfassend kann gesagt werden:

1. Simulationsprogramme sind zu komplex, um technisches Denken und Handeln zu erlernen. Sie fördern den Umgang mit dem System und sind nur für spezielle Aufgabenstellungen geschrieben worden.
2. Systemmodellierungs-Tools erfordern das Erlernen einer eigenen Sprache, die später nicht auf andere Problemlösungsverfahren übertragbar ist.
3. Systemmodellierungs-Tools sind nicht beliebig erweiterbar.
   (vergl. [Bresges, 2002])

6.2 Die SiMo-Umgebung

Die **SiMo-Umgebung** (Simulations- und Modellierungs-Umgebung) ist ein Teilergebnis dieser Arbeit. Sie beinhaltet ansatzweise die Möglichkeiten professioneller Gebäudesimulationssoftware. Im Folgenden wird das im Rahmen dieser Arbeit entwickelte Systemmodell eines Hauses kurz erläutert, um die Vor- und Nachteile dieser spezifizierten Modellbildung aufzuzeigen.

Das zu erstellende technische System ist ein **Haus**, das stark vereinfacht aus Räumen, Wänden und Heizgeräten besteht.
Die **Umgebung**, alles außerhalb der Außenwände, wird als spezieller Raum mit unendlichem Volumen betrachtet. Dies bedeutet praktisch, dass sich seine Temperatur nicht ändert und somit konstant bleibt. Die Temperatur kann nur über die Eingabe des Users während der Simulationslaufzeit geändert werden.
Der **Raum** wird als Wärmepotenzial betrachtet. Die Raumtemperatur wird im ganzen Raum als homogen angenommen. Die Wärmekonvektion im Raum wird vernachlässigt[5].
Die **Wand** wird als Wärmewiderstand verstanden, ohne kapazitive Eigenschaften. Einer Wand wird ein Wärmedurchgangskoeffizient **U** zugeordnet, der in unserem Modell als temperaturunabhängig angenommen wird.

5 nicht die der Wände oder Heizgeräte

Das **Heizgerät** wird als Wärmequelle für einen Raum verstanden. Es gibt verschiedene Typen, die unterschiedliches Verhalten bei der Wärmeabgabe zeigen. (vergl. Kapitel 7.3.3) Gemeinsam haben die Heizgeräte die Eigenschaft, dass die an den Raum abgegebene Wärmeenergie sofort dem ganzen Raum angerechnet wird. Der Wärmeübergangskoeffizient kann je nach Gerätetyp berücksichtigt werden.
Die **Wärmestrahlung** wird für das ganze Modell außer Acht gelassen.

6.3 Gegenüberstellung der SiMo-Umgebung mit gewerblichen Wärmebedarfs- und Gebäudesimulationsprogrammen

Betrachtet man dieses einfache Modell mit bestehenden Wärmebedarfsprogrammen, so kann festgestellt werden, dass sich dieses einfache Modell nicht für professionelle Berechnungen eignet. Die Berechnungen des Wärmebedarfs für unterschiedliche Gebäude sind in der EN DIN 12831 Berechnung des Wärmebedarfs von Gebäuden. Auf dieser Grundlage bewegen sich die Berechnungen der professionellen Software. In diesen Softwarepaketen stehen dann auch unter anderem Bibliotheken und Tabellen mit Kenndaten von Baustoffen, Heizgeräten, usw. zur Verfügung. Diese Berechnungsprogramme sind für die einfache und schnelle Abwicklung von Aufträgen in der Industrie ausgelegt. Ein Lernen der Systemzusammenhänge ist mit dieser Software schwierig, wenn nicht sogar gänzlich unmöglich. Sie ist als Hilfsmittel für den Fachmann gedacht (vergl. [mhSoft]).

Es handelt sich bei diesen Programmen nicht um Simulationen im eigentlichen Sinne. Es werden keine dynamischen Prozesse dargestellt, sondern hauptsächlich Berechnungen durchgeführt, die auf Erfahrungs- und Mittelwerten, die in Normen festgeschrieben sind, beruhen.

Eine Alternative wären Gebäudesimulations-Programme, die zum Ziel haben, Gebäude dynamisch zu betrachten. Sie haben größtenteils auch den objektorientierten Ansatz, wie er in dieser Arbeit verwendet wird (vergl. [SMILE]).

Das Modellierungs-Tool **SMILE**[6] wird in dieser Arbeit nicht verwendet, weil es auf Gebäudesimulationen spezialisiert ist. Das Konzept in dieser Arbeit wird zwar anhand des Beispiels **Haus** vorgestellt, ist aber auf jedes andere beliebige technische System übertragbar. Es ist möglich, das in dieser Arbeit vorgestellte Hausmodell den Hausmodellen, die mit SMILE erstellt werden können, gleichwertig anzupassen, ohne das Konzept zu verlassen. Es muss lediglich durch genauere Beschreibungen der einzelnen Subsysteme erweitert und konkretisiert werden.

6 Um nur ein Beispiel zu nennen (vergl. [SMILE])

7 Entwicklung der Simulations- und Modellierungs Umgebung *HausSim*

In diesem Kapitel wird beschrieben, wie eine Simulations- und Modellierungs-Umgebung (**SiMo-Umgebung**) unter Anwendung der *Systemtheorie der Technik* entwickelt werden kann.

In Kapitel 7.1 wird anhand der Systemtheorie nach Ropohl ein abstraktes Modell des technischen Systems **Haus** entwickelt. Dieses abstrakte Modell wird mit Hilfe der drei Aspekte der *Systemtheorie der Technik* konkretisiert.
Kapitel 7.2 überführt das systemtheoretische Modell in die Beschreibungssprache **UML**. Hier werden die einzelnen Subsysteme in Klassendiagrammen näher konkretisiert.
Kapitel 7.3 setzt das aus Kapitel entwickelte abstrakte Modell in die Programmiersprache **Java** um. Dabei werden die Objekte mit ihren Eigenschaften und Methoden genau beschrieben. Die Objekte werden dann in die SiMo-Umgebung eingebettet.
Der Weg der Konkretisierung wird in Abb. 3 dargestellt. Die allgemeinen transformations Vorschriften verdeutlicht Abb. 4. und Abb. 5 gibt einen Überblick über die hierarchische Struktur der **SiMo-Umgebung**.

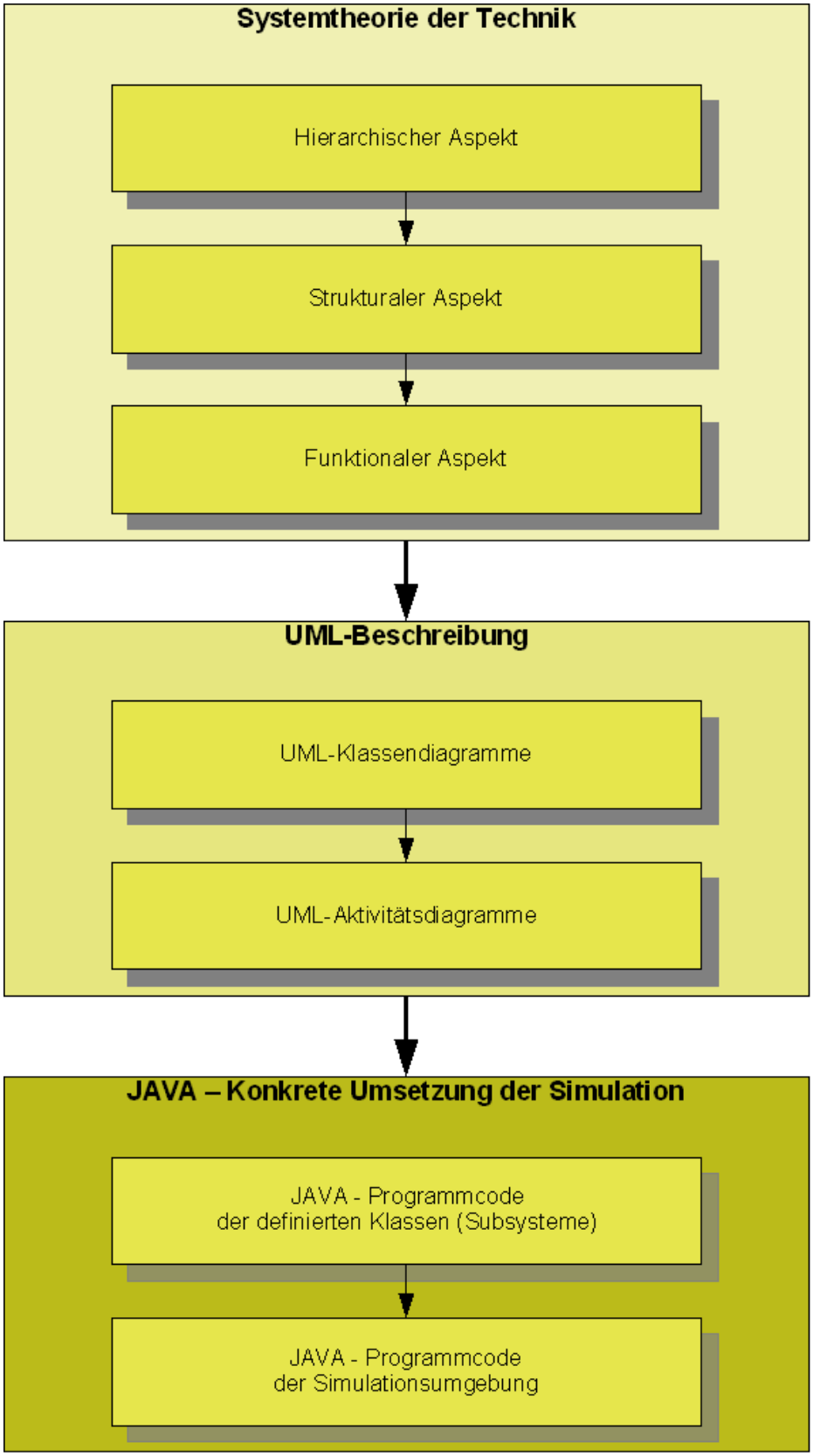


*Abb. 3 Stufen der Konkretisierung des Systems **Haus***

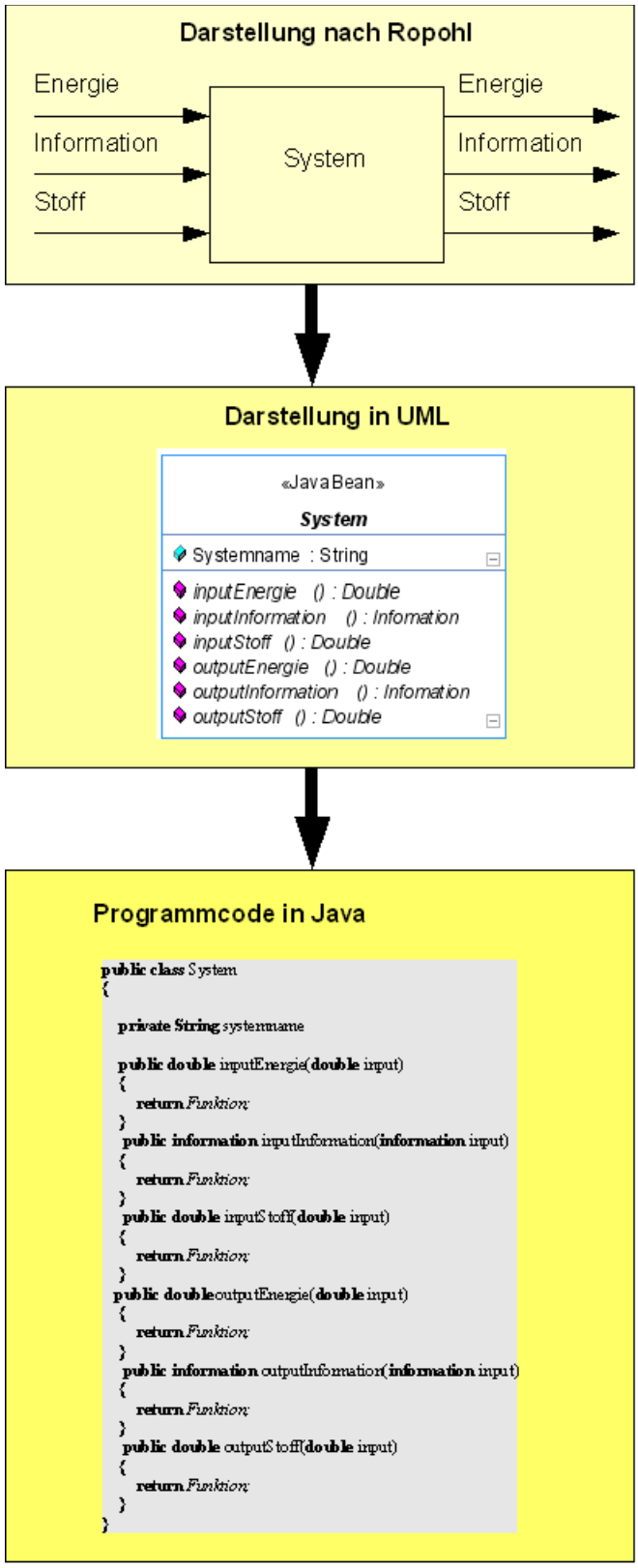


Abb. 4: Allgemeine Transformationsvorschriften für das Systemmodell

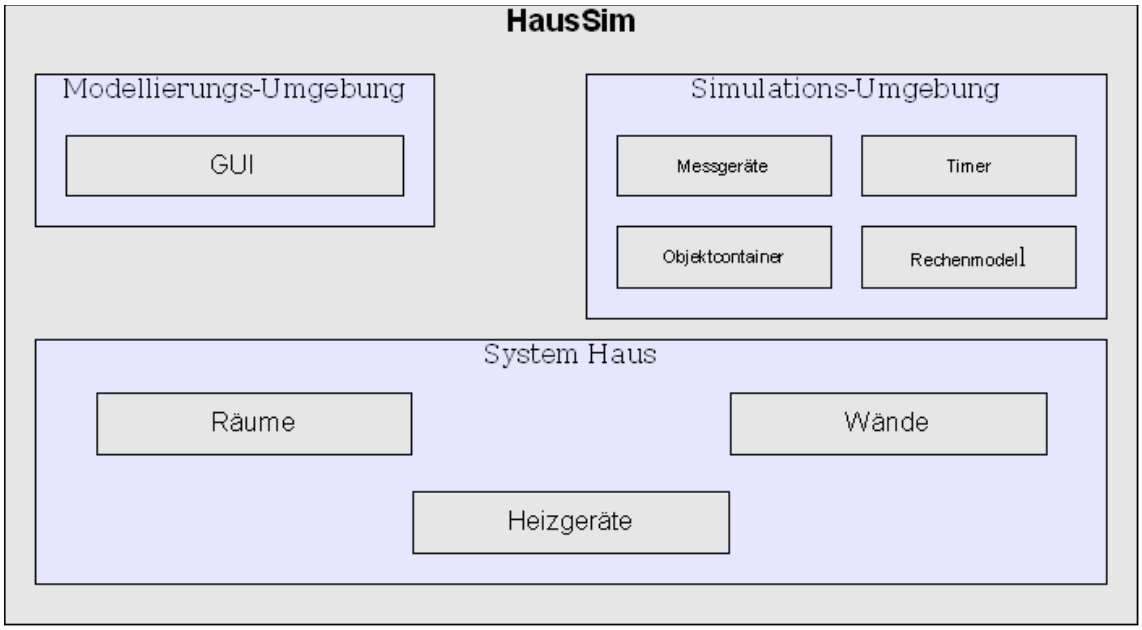


*Abb. 5 Hierarchische Struktur der SiMo-Umgebung **HausSim***

7.1 Erstellung eines Modells des technischen Systems *Haus* auf der Grundlage der *Systemtheorie der Technik*.

Gemäß der *Systemtheorie der Technik* lassen sich alle Systeme unter drei verschiedenen Aspekten darstellen. Das hier zu analysierende System soll das technische System **Haus** sein. Hier wird zunächst einmal der hierarchische, der strukturale und der funktionale Aspekt betrachtet.

7.1.1 Der *hierarchische Aspekt* des Systems *Haus*.

Das System **Haus** besteht aus verschiedenen Subsystemen. Diese Subsysteme sind die einzelnen Räume. Räume wiederum beinhalten ihrerseits wieder Subsysteme, und zwar die Wände und Heizgeräte.
Prinzipiell wäre es möglich auch diese Subsysteme bis ins kleinste Detail zu zergliedern und aus diesem fein gegliederten Systemmodell die Simulation zu erstellen. Dies ist unter anderem eine der Stärken des Gesamtkonzeptes.
Im Rahmen dieser Arbeit soll jedoch der Übersichtlichkeit halber die vorgenannte Abstraktionsebene beibehalten werden, die Hierarchie der Systeme im System **Haus** zu erfassen.

Ein System **Haus** ohne eine Umgebung ist wirklichkeitsfremd. Deshalb soll die Systemgrenze des Systems **Haus** noch erweitert werden, und zwar indem die Umgebung in der Betrachtung berücksichtigt wird. Das Hauptsystem **Haus** wird nicht umbenannt, da die Umgebung nicht aktiv an den Abläufen im System **Haus** beteiligt ist, sondern nur Einfluss im Sinne einer Störung auf das System **Haus** nimmt. Diese Störung ist dennoch so fundamental für die Abläufe im System **Haus**, dass die Umgebung später als Subsystem integriert wird. Dies wird im Weiteren detailliert dargelegt.[7] Durch die dickere Strichstärke des Kästchens für das System **Haus** soll hervorgehoben werden, dass das System **Haus** das Hauptsystem ist, welches mehrere Subsysteme **Raum** beinhalten und diese jeweils wieder **Wände** und **Heizgeräte**. Die Umgebung wird somit zum Supersystem (s. Abb. 6).

7 Siehe Kapitel 13.2.2

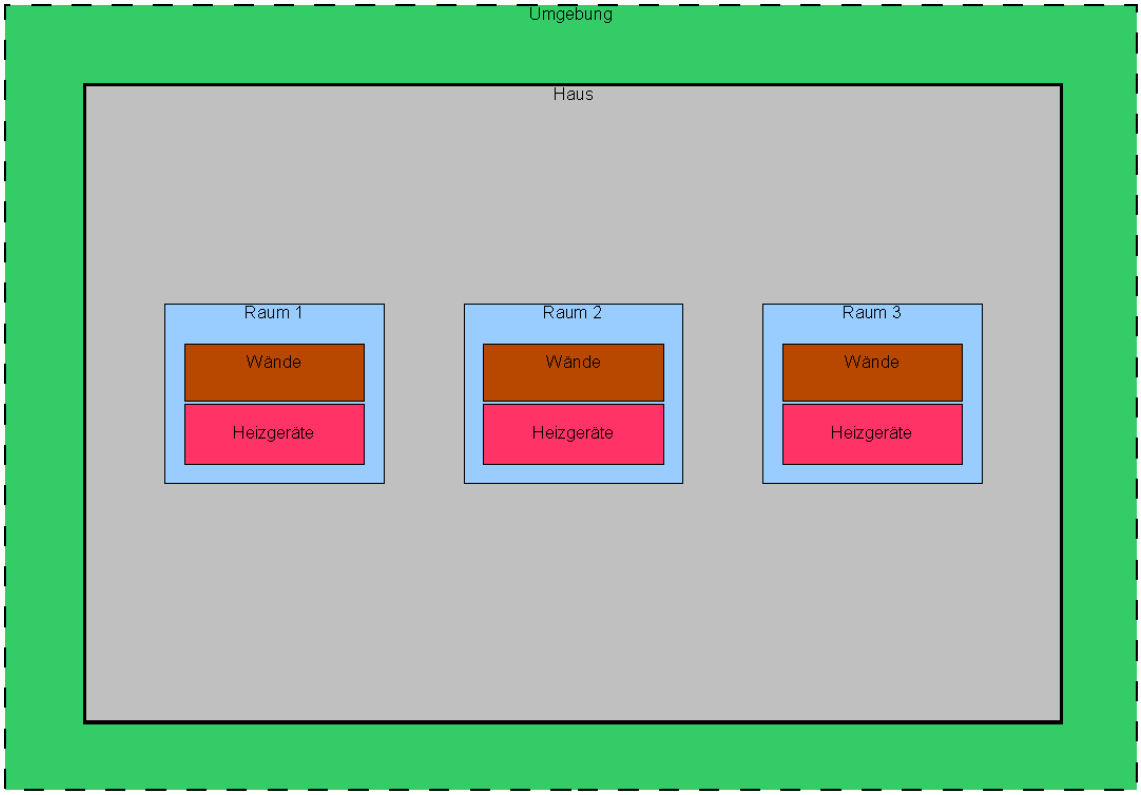


Abb. 6 Hierachischer Aspekt des Systems ***Haus***

Bis jetzt können noch keine Aussagen erfolgen, wie die einzelnen Subsysteme miteinander interagieren. Dies wird mit Hilfe des *strukturalen* und des f*unktionalen Aspektes* möglich.

7.1.2 Der *strukturale Aspekt des* Systems *Haus*

Der *strukturale Aspekt* beschreibt nun die Relationen der Subsysteme untereinander.

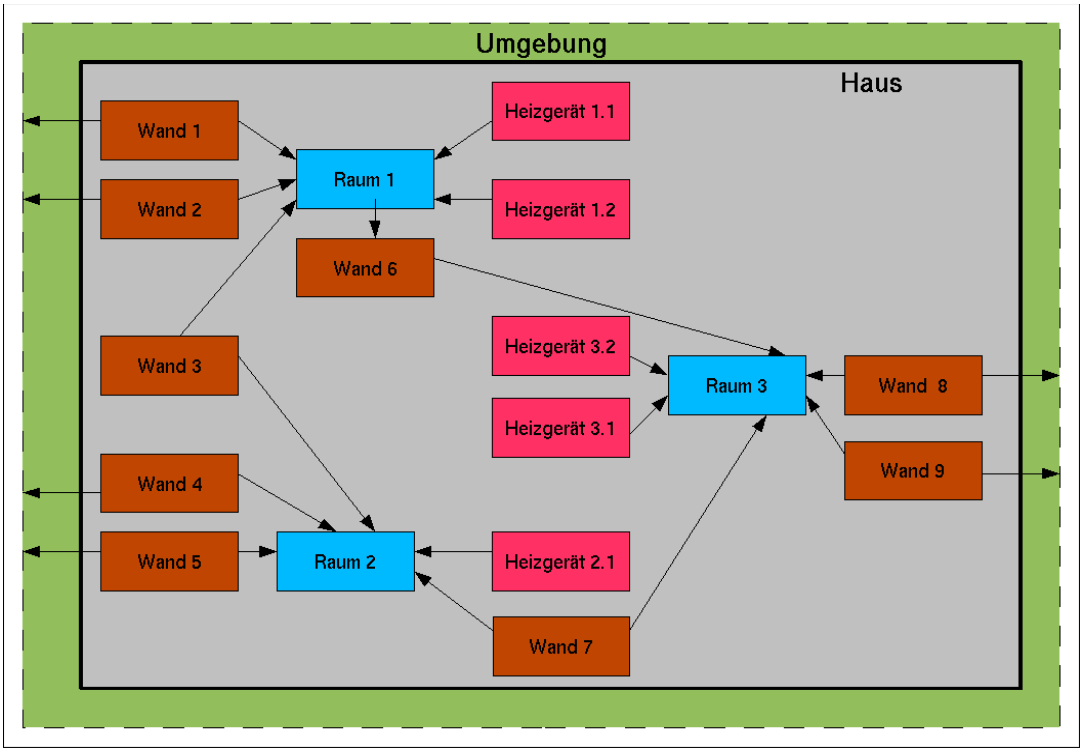


Abb. 7 Strukturaler Aspekt des Systems Haus

Wie aus Abb. 7 ersichtlich ist, beinhaltet das System **Haus** Subsysteme der Kategorien Räume, Wände und Heizgeräte. Die Subsysteme Wände wurden nun in weitere Subsysteme **Wand n** zerlegt. Ebenso wurde mit den Heizgeräten verfahren. Es besteht also jeweils eine n:1-Beziehung zwischen **Wände–Räume** und **Heizgeräte-Räume**. Dies bedeutet, dass einem Element, z. B. einem Raum aus der Liste **Räume**, mehrere Elemente, z. B. eine Wand oder ein Heizgerät aus den Listen **Wände** und **Heizgeräte**, zugeordnet werden können. Die Pfeile, die die Richtung der Zuordnung symbolisieren, zeigen immer in Richtung der Räume. Weiter wird ersichtlich, dass jedes Heizgerät nur einem Raum zu-

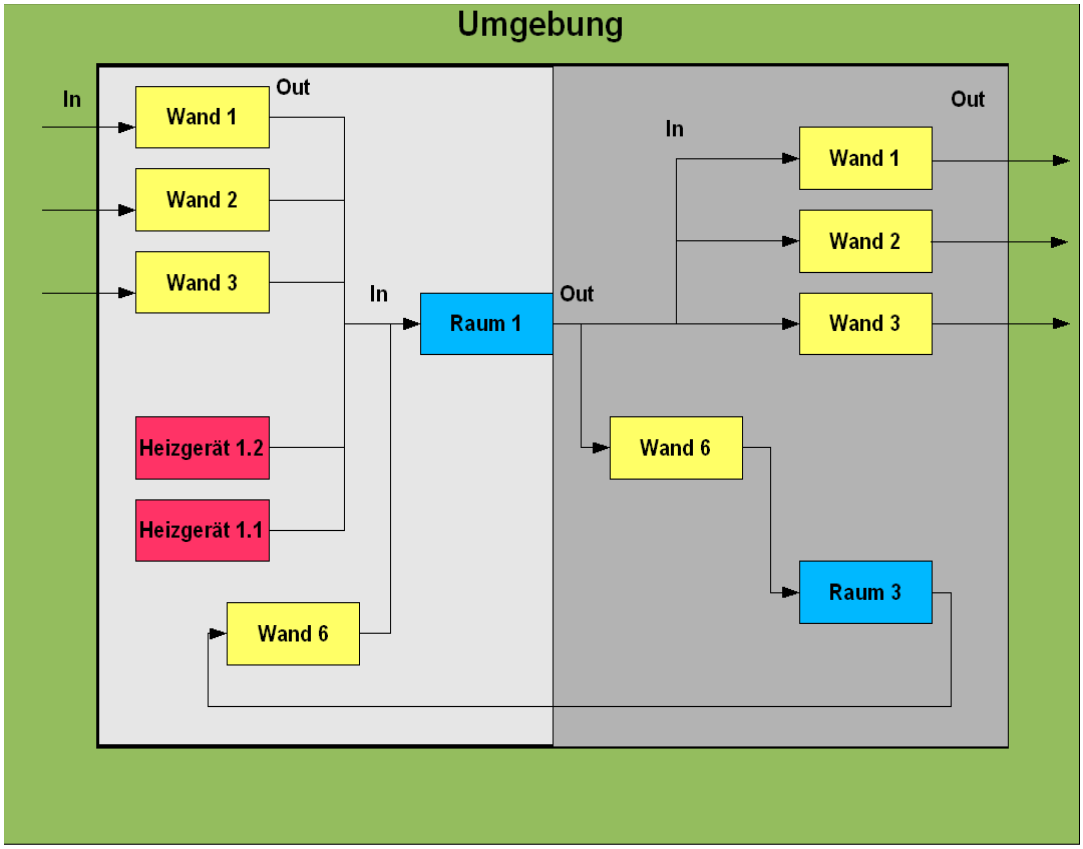
geordnet wird. Dies ist bei der Wand nicht der Fall, sie wird immer genau zwei Räumen zugeordnet. Dabei wird die Umgebung auch als Raum betrachtet. Die Erläuterung zu diesem Sachverhalt wird in Unterkapitel 7.2.1 gegeben.

Einige Räume sind scheinbar austauschbar, weil sie sich nach dem Informationsgehalt der Abb. 6 nur durch den Namen unterscheiden. Dies zeigt, dass das System mit den beiden Aspekten noch nicht vollständig erfasst werden kann. Denn die beiden Wände können sich in den Eigenschaften unterscheiden. **Wand 1** kann einen anderen Wärmedurchgangskoeffizienten haben als **Wand 2** (z. B. **Wand 1** besteht aus Steinen, **Wand 2** ist eine Fensterfront). So entstehen zwei verschiedene Elemente mit den gleichen Zuordnungen.

7.1.3 Der *funktionale Aspekt* des Systems *Haus*.

Wird nun das System **Haus** unter dem *funktionalen Aspekt* betrachtet, werden die Beschreibungen konkreter. Den Systemen werden jetzt Inputs und Outputs zugeordnet.

In Abb. 8 wird der *funktionale Aspekt des* Systems **Haus** dargestellt. Aus Gründen der Übersichtlichkeit wird nur die Funktionalität um **Raum 1** herum betrachtet. Es ist ersichtlich, dass die Subsysteme über die Inputs und Outputs miteinander verknüpft sind. Die Subsysteme interagieren über ihre Schnittstellen Input und Output. Es können zwei Subsystemtypen unterschieden werden und zwar gibt es Subsysteme mit nur einem Output (**Heizgeräte**) und Subsysteme mit einem Input und einem Output (**Wände** und **Räume**). Diese Inputs und Outputs können aber auch mit mehreren Outputs verbunden sein, so dass man unter **einem** Input oder Output mehrere zusammenfassen kann. Die Inputs des Raums 1 sind mit den Outputs der Subsysteme **Wand 1,2,3,6** und den **Heizgeräten 1.2, 1.1** verbunden.



Nach der *Systemtheorie der Technik*, werden die drei Input- und Outputkategorien Stoffumsatz, Energieumsatz und Informationsumsatz unterschieden.

Dieses Modell des Hauses wird in die Systeme des **Energieumsatzes** eingeordnet, weil es die Wämeverteilungen und -strömungen des Hauses abbilden soll. Somit zeigen die Verknüpfungen in Richtung des Energieflusses. Einige Wände erscheinen doppelt, damit die Darstellungsweise übersichtlich bleibt. Korrekterweise müsste der Output des Raumes mit jedem Input der Wände auf der linken Seite mit einer Verknüpfung verbunden werden. Dies würde jedoch fälschlicherweise den Eindruck einer Rückkopplung erwecken. Die gewählte Darstellungsweise soll jedoch zeigen, dass die Wände Inputs und Outputs haben, und dass die Energieflüsse sowohl von der Umgebung durch die Wand in den Raum als auch vom Raum durch die Wand in die Umgebung möglich sind. Mit Hilfe dieser Darstellungen der Perspektive wird die Abhängigkeit der verschiedenen Subsysteme untereinander deutlich.

Die Heizgeräte haben nur Outputs, sie erzeugen scheinbar Energie. Diese Betrachtungsweise ist hier legitim, da in diesem Modell nicht betrachtet werden soll, wie die Wärmeenergie in das Heizgerät gelangt, sondern nur welchen Einfluss die Energieabgabe an den Raum auf das gesamte System hat.

Räume, **Wände** und die **Umgebung** sind Systeme mit Inputs und Outputs. Wie die Input- und Outputgrößen nun in den Systemen behandelt werden, geht jedoch aus diesen Betrachtungsweisen nicht hervor. Dazu dient die Definition der einzelnen Systeme oder Elemente.

7.2 Transformation und Konkretisierung des technischen Systems *Haus* nach UML

Dieses Kapitel beschreibt, wie das abstrakte Objekt **Haus** zu konkretisieren und die einzelnen Subsysteme in **UML** (Unified Modeling Language) zu transferieren sind. Zuerst werden die Objekte jeweils allgemein definiert und dann als Klassendiagramme in UML abgebildet. Dabei wird deutlich werden, wie ähnlich sich die Darstellung eines Objekts in UML und die Darstellung mittels der Symbolik der Systemtheorie der Technik sind.

7.2.1 Definition des Objektes Raum

Nun wird das Objekt **Raum** definiert. Es werden die konkreten Eigenschaften und Methoden des Raums definiert. Für ein einfaches Raummodell, welches den **Raum** als ein Wärmepotenzial betrachtet, lassen sich die Eigenschaften in einer Tabelle zusammentragen.

Eigenschaft	***Bemerkung***
Name	Bezeichnung des Raumes (Identifikation)
Höhe - H	Höhe des Raumes
Breite - B	Breite des Raumes (x-Koordinate)
Tiefe - T	Tiefe des Raumes (y- Koordinate)
Volumen - V	Volumen des Raumes (ohne Wände)
Temperatur - ϑ	Temperatur im Raum
Masse - m	Masse des Mediums im Raum (ohne Wände)
Spezifische Wärmekapazität - c	Spezifische Wärmekapazität des Mediums im Raum
Dichte - ρ	Dichte des Mediums im Raum
Wärmeinhalt - Q	Wärmeinhalt des Raums (des Mediums im Raum)

Da der Raum ein Wärmepotenzial beschreibt, hat er keine Methoden, die **fremde** Objekte manipulieren könnten. Somit lassen sich zunächst einmal nur zwei abstrakte Methoden angeben, die für den Input und den Output des Raumobjektes stehen.

Der Raum hat jedoch eine *innere* Funktion. Er kann die Energie, die ihm über den Input zugeführt wird, speichern. Das heißt, er erhöht sein Energie-, oder konkreter, sein Wärmepotenzial. Diese Funktion soll **steigernWärmepotenzial ()** heißen.

Methoden des Raumes

Methodenname	***Parameter***	***Bemerkung***
inputEnergie()	k. A.	repräsentiert zunächst einmal den Input des Objektes **Raum**.
outputEnergie()	k. A.	repräsentiert zunächst einmal den Output des Objektes **Raum**.
steigernWärmepotenzial	deltaWärmemenge	Diese Funktion soll dafür sorgen, dass über die Inputs zugeführte Energie der Gesamtenergie des Raumes hinzugefügt wird, und dass abgeführte Energie über die Outputs der Gesamtenergie abgezogen wird

Das Objekt **Raum** lässt sich nun in **UML** in einem Klassendiagramm symbolisch darstellen: Abb.9

Die Typen hinter den Eigenschaften (name, breite, usw.) und den Methoden (inputEnergie(), outputEnergie(), usw.) des Objektes geben den Datentyp (string (Zeichenkette), double (rationale Zahl doppelter Genauigkeit), usw.) an.

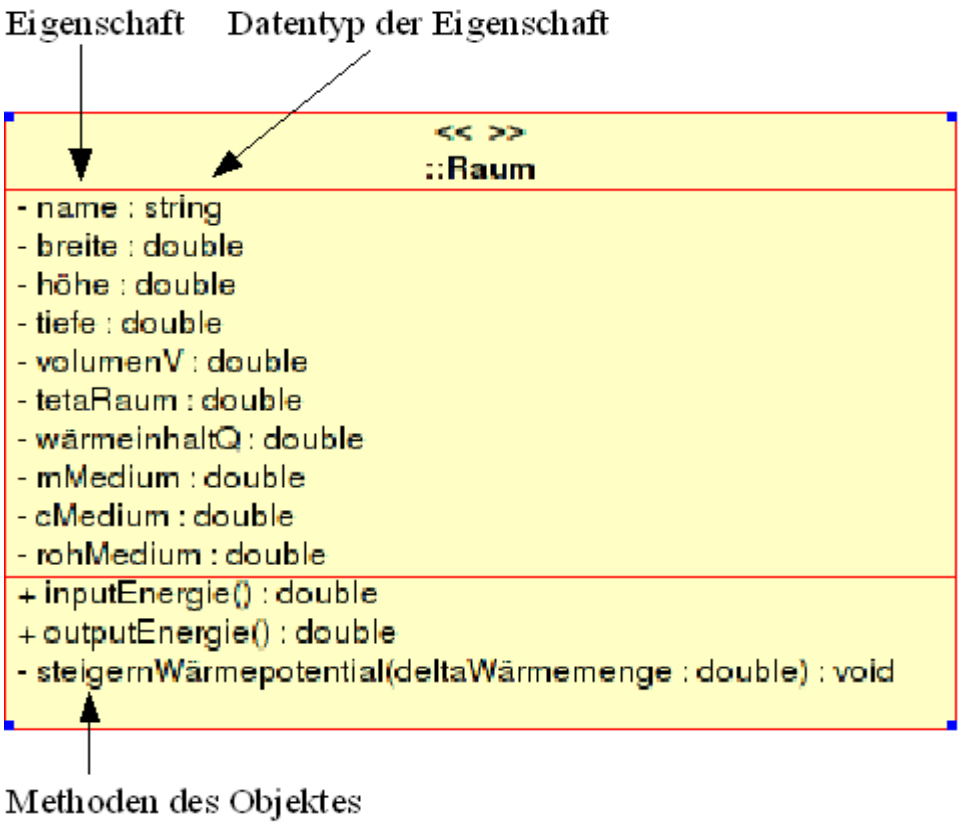


Abb. 9 Raumobjekt in UML-Darstellung

In Kapitel 7.1.1 wurde bereits erwähnt, dass die **Umgebung** auch als **Raum** betrachtet werden soll. Wo liegen nun die Gemeinsamkeiten und die Unterschiede zwischen diesen beiden Objekten? Beide Objekte stellen ein Wärmepotenzial dar, welches über die Temperatur erfasst werden kann. Beiden Objekten kann somit Wärmeenergie zugeführt oder abgeführt werden. Der Unterschied liegt im Volumen. Bei den Räumen innerhalb des Hauses ist das Volumen begrenzt. Dagegen hat die Umgebung für unsere Betrachtungen ein un-

endliches Volumen. Das bedeutet für das Zuführen und Abführen von Wärmeenergie, dass sich dies im Wärmepotenzial der Umgebung nicht bemerkbar macht. Es kann somit vernachlässigt werden. Daraus ergibt sich, dass die Eigenschaften, die das Wärmepotenzial über das Medium und das Volumen berechnen, für die Umgebung irrelevant sind. Das Wärmepotenzial der Umgebung lässt sich alleine über die Temperatur der Umgebung definieren.

7.2.2 Definition des Objektes Wand

Die Wand kann als ein Objekt betrachtet werden, das als Wärmewiderstand fungiert. In diesem Modell werden die wärmekapazitiven Eigenschaften einer Wand vernachlässigt. Es wird davon ausgegangen, dass die Wand sich so verhält, als ob die angrenzenden Wärmepotenziale der Räume schon immer bestanden haben, und der Temperaturgradient in der Wand somit stabil ist. Das bedeutet, die Wand ist in einem stationären Zustand und wirkt somit nur noch als Wärmewiderstand.

Unter diesen Voraussetzungen lassen sich der **Wand** folgende Eigenschaften zuordnen.

Eigenschaft	***Beschreibung***	***Bemerkung***
Name	Name der Wand	Identifikation der **Wand**
Breite – B	Breite der Wand	
Höhe – H	Höhe der Wand	
Fläche - A	Fläche der Wand	
U-Zahl	Wärmedurchgangskoeffizient	
Temperatur Raum 1	Temperatur Raum 1	
Temperatur Raum 2	Temperatur Raum 2	
Wärmefluss dQ/dt	Wärmefluss durch die Wand	

An den Eigenschaften wird deutlich, dass die Wand zwei Räumen zugeordnet werden muss. Dies entspricht den Relationen entsprechend dem strukturalen Aspekt des Systems **Raum**.

Unter dem funktionalen Aspekt bekommt das Objekt **Wand** eine Wärmemenge angeboten, die es am Input in sich aufnehmen kann. Durch die Funktionalität der **Wand** als Wärmewiderstand wird verhindert, dass die vollständige Wärmemenge des Raumes auf einmal aufgenommen wird. Somit wird der Wärmefluss in das System **Wand** durch die Funktion **wärmefluss()** begrenzt. $\Delta \frac{Q}{\Delta t} = UA(\vartheta_{Raum1} - \vartheta_{Raum2})$. Die aufgenommene Wärmeenergie wird in der Wand selbst nicht mehr verändert, sondern direkt zum Output der Wand geleitet. Somit sind aufgrund des stationären Zustandes der Wand der Input und Output synchron.

Anmerkung: *An diesem Modell wird der Unterschied zwischen technischem Systemdenken nach Ropohl und einer naturwissenschaftlichen Betrachtungsweise deutlich. Der Physiker würde an dieser Stelle anmerken, dass die Wand passiv ist und keine Funktion hat. Er würde die Potenzialdifferenz der beiden Räume als Ursache für den Wärmefluss benennen. Somit hätte er drei passive Objekte definiert, und zwar zwei Räume mit jeweils unterschiedlicher Temperatur (Potenzialdifferenz) und die Wand als Widerstand. Es fehlt jetzt ein aktives Objekt, damit ein Wärmeausgleich stattfinden kann. Hier müsste er dann ein Objekt* ***Natur*** *schaffen, das keine Eigenschaften besitzt, sondern nur Methoden, die anhand der Eigenschaften der passiven Objekte die Naturgesetze wirken lassen. Dieses Objekt würde nun die Eigenschaften der anderen Objekte dementsprechend manipulieren. Es würde anhand der Wandeigenschaften und der Raumtemperaturen den Wärmefluss berechnen, und auf dieser Grundlage die Eigenschaft* ***Wärmeinhalt*** *der Räume ändern. Die Eigenschaften der Wand würden bei unserem vereinfachtem Modell unangetastet bleiben.*

Für das Objekt **Wand** werden folgende Funktionen definiert.

Funktion/Methode	***Parameter***	***Bemerkung***
inputEnergie()	Temperaturen Raum1 und Raum 2 (werden über den **InputInformation** eingelesen)	Der Input wird über die Methode **wärmefluss()** geregelt. Die Funktion **wärmefluss()** benötigt jedoch die Information der beiden Raumtemperaturen. Diese werden über den Informations-Input eingelesen, obwohl es sich hier nicht um den klassischen Informationstransport handelt. (Das Problem wird in Kapitel 7.3.2 ausführlich erläutert)
outputEnergie()	gleich dem Input	
wärmefluss()	Raumtemperatur von Raum1 und Raum2	Die Parameter werden von der Funktion Input-Information an diese Funktion übergeben. Diese Funktion tritt nicht nach außen hin auf. Sie ist *„privat“.*

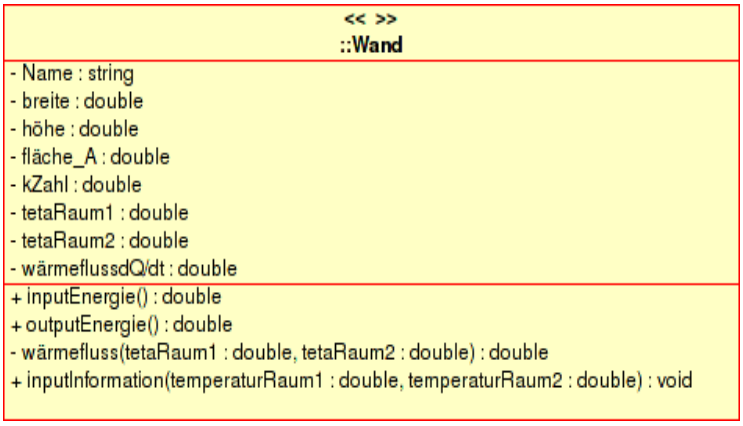


Abb. 10 Darstellung des Wandobjektes in UML

Somit wäre das Objekt **Wand** definiert. Die Darstellung in **UML** wird in Abb. 10 gezeigt.

7.2.3 Definition des Objektes Heizgerät

Das **Heizgerät** ist ein Objekt mit der Funktion dem Raum Wärme zuzuführen. Dies lässt sich durch verschiedene Heizgerätemodelle realisieren. In dieser Arbeit werden zwei recht unterschiedliche Modelle vorgestellt. Sie zeigen, dass die Modelle nach Belieben erweitert werden können. Je nach Anspruch an die Realitätsnähe können zusätzlich mathematische Modelle eingefügt werden.

Heizgerät Typ I ohne Regelung

Dieser Typ hat eine simple Funktionalität. Er soll, wenn er aktiv ist, ständig eine vorgegebene konstante Wärmemenge in den Raum leiten. Für diesen Typ wird nur eine geringe Anzahl von Eigenschaften benötigt.

Eigenschaft	***Beschreibung***	***Bemerkung***
Name	Name des Objektes	
Wärmeleistung	Wärmeleistung der Heizung in Betrieb	

Eigenschaft	***Beschreibung***	***Bemerkung***
Power ON/OFF	Zustand der Heizung **Ein** oder **Aus**	

Ebenso sind die Funktionen des Heizgerätes auch beschränkt.

Funktion	***Parameter***	***Beschreibung***
inputEnergie()		nicht belegt. (siehe funktionale Perspektive)
outputEnergie()		gibt die Energie an den Raum ab.
inputInformation()	Funktion Power und Parameter On/Off	Flag, mit den Werten ***true*** und ***false***
wärmeabgabe()	Power	Der Parameter Power entscheidet, ob der Output den Wert Wärmeabgabe erhält oder nicht.

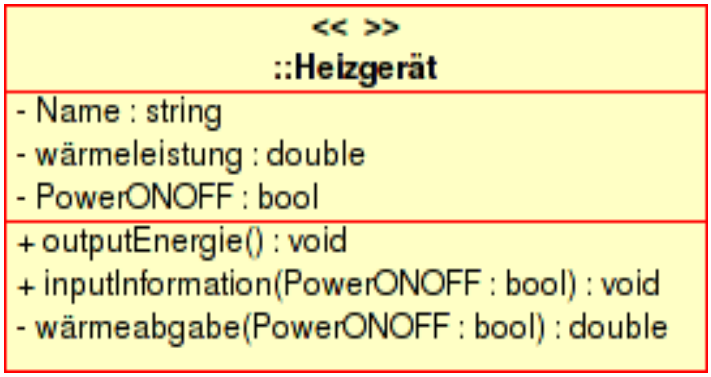


Abb. 11 Darstellung des Objektes Heizgerät in UML

Heizgerät Typ I mit Regelung

Für die Regelung werden den Eigenschaften des Heizkörpers die Eigenschaften **Raumtemperatur max** und **Raumtemperatur min** hinzugefügt, die das Ein- und Ausschalten der Heizung regeln. Dem Raum wird also nur noch eine konstante Wärmemenge zugeführt, solange die Raumtemperatur unter dem angegebenen Wert liegt. Somit lassen sich Sachverhalte darstellen, bei denen es nicht auf die exakte Abbildung eines realen Heizgerätes im Raum ankommt.

Eigenschaft	***Beschreibung***	***Bemerkung***
Name	Name des Objektes	
Wärmeleistung	Wärmeleistung der Heizung, in Betrieb	
Power ON/OFF	Zustand der Heizung **An** oder **Aus**.	
Raumtemperatur max	Raumtemperatur, bei der das Heizgerät abschalten soll	

Eigenschaft	*Beschreibung*	*Bemerkung*
Raumtemperatur min	Raumtemperatur, bei der das Heizgerät einschalten soll	

Zusätzlich benötigt das Objekt Heizgerät eine Funktion (Methode), die die Regelung übernimmt. Das Objekt sieht nun wie unten in UML dargestellt aus (s. Abb.12).

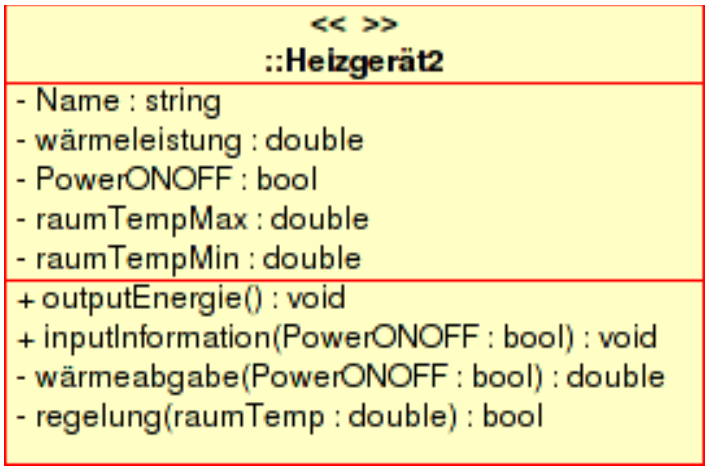


Abb. 12 Darstellung des Objektes Heizgerät Typ I mit Regelung in UML

Heizgerät Typ II ohne Regelung

Der Heizgerät Typ II soll ein Heizgerät simulieren, bei dem berücksichtigt wird, dass es einen Wärmespeicher hat, und dass die Wärmemengenabgabe an den Raum von der Temperaturdifferenz zwischen der Heizgeräteoberfläche und der Raumtemperatur abhängt. Als Vorbild soll ein Konvektor dienen. Der Konvektor ist mit einer bestimmten Menge Wasser gefüllt. Über eine bestimmte Fläche gibt er Wärme an den Raum ab. Es wird weiter davon ausgegangen, dass das Material des Konvektors die Eigenschaft besitzt einen hohen Wärmeleitkoeffizienten zu haben, wodurch die Näherung zulässig wird, die Temperatur

der Oberfläche des Gerätes mit der des Wassers im Inneren gleichzusetzen. Somit hängt der Wärmefluss vom Konvektor in den Raum lediglich vom Wärmeübergangskoeffizienten der Konvektoroberfläche ab.

Die neue Eigenschaftentabelle sieht wie folgt aus:

Eigenschaft	***Beschreibung***	***Bemerkung***
Name	Name des Objektes	
Power ON/OFF	Zustand der Heizung **ein** oder **aus**	
Mediummasse m	Masse des Mediums im Heizkörper	
spezifische Wärmekapazität c	Medium im Heizkörper	
Fläche A	effektive Fläche, über die die Wärme an den Raum abgegeben wird.	
Wärmeübergangskoeffizient α	der Wärmeübergangskoeffizient an der Heizkörperoberfläche	
Raumtemperatur ϑ	die aktuelle Raumtemperatur	
Raumtemperatur max	Raumtemperatur, bei der das Heizgerät abschalten soll	
Raumtemperatur min	Raumtemperatur, bei der das Heizgerät einschalten soll	

Dem Objekt müssen keine weiteren Methoden hinzugefügt werden. Die Methode **wärmeabgabe()** muss lediglich an die neuen Berechnungen angepasst werden.

Somit sieht das neue Objekt für den Heizgerät Typ II so aus: (s. Abb. 13)

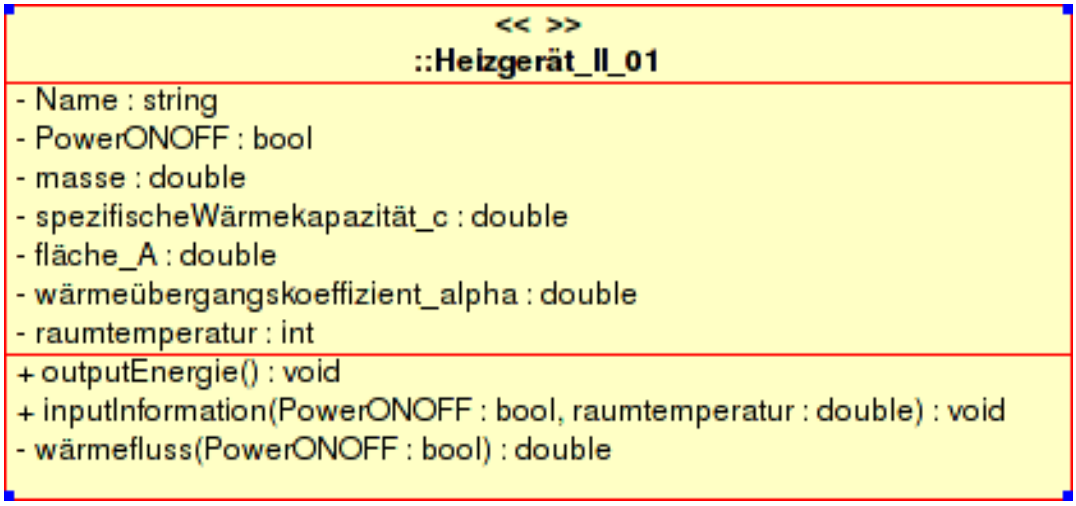


Abb. 13 Darstellung des Objektes Heizgerät Typ II ohne Regelung in UML

Die Raumtemperatur hat für das Verhalten dieses Heizgerätes ein weitere Bedeutung. Sie dient nicht nur als Regelgröße für die Regelung, sondern ist auch für die Berechnungen der Wärmeabgabe an den Raum notwendig.

Die Funktionsweise dieses Heizkörpers ändert sich auf Grund der neuen Funktionalität der Methode **wärmefluss()**. Wenn der Heizkörper eingeschaltet ist und heizt, soll das Wasser über die gesamte Fläche die gleiche Temperatur (Vorlauftemperatur) haben. Wird der Heizkörper ausgeschaltet, nimmt die Temperatur stetig ab, d. h. die Wärmemenge, die an den Raum abgegeben wurde, fehlt dem Wärmeinhalt des Heizkörpers. Diese Temperatur, die sich aus dem neuen Wärmeinhalt des Heizgerätes ergibt, soll Rücklauftemperatur genannt werden. Sie ist zwar keine Rücklauftemperatur im eigentlichen Sinne, weil das Wasser im Heizkörper steht und nicht abfließt, aber sie ist die gleiche Temperatur, mit der das Wasser aus dem Heizkörper herausfließen würde und als Rücklauftemperatur angegeben würde, wenn der Heizkörper aktiv ist. Dies beschreibt eine gute Näherung an das Verhalten eines realen Wasserheizkörpers. Der Schwachpunkt dieses Modells liegt in der Aufwärmphase des Heizkörpers. In der Realität hat die gesamte Heizkörperoberfläche nicht gleich nach dem Einschalten die Temperatur des Vorlaufs, sondern es vergeht ein gewisser Zeitraum, bis die gesamte Fläche die maximale Temperatur erreicht hat. Außerdem hat jede Heizfläche einen Temperaturgradienten, so dass von einer mittleren

Temperatur des Heizkörpers gesprochen werden muss. Da es aber in dieser Arbeit nicht um die möglichst naturgetreue Modellierung eines Heizkörpers geht sondern um das Prinzip einer Modellbildung, wird diese Ungenauigkeit toleriert.

Mit diesem Modell des Heizgerätes können nun auch Effekte des Überheizens simuliert werden, und es können die Einflüsse der Wärmekapazität des Wassers auf die Raumtemperatur beobachtet werden.

Heizgerät Typ II mit Regelung

Dieser Heizgerät Typ II ist mit einer Regelung erweitert worden. Die zusätzlichen Eigenschaften und Methoden sind die gleichen wie in Heizgerätetyp I mit Regelung.

Es ergeben sich folgende Eigenschaften und Funktionen.

Eigenschaft		
Name	Name des Objektes	
Power ON/OFF	Zustand der Heizung **Ein** oder **Aus**	
Mediummasse m	Masse des Mediums im Heizkörper	
spezifische Wärmekapazität c	c vom Medium im Heizkörper	
Fläche A	effektive Fläche, über die die Wärme an den Raum abgegeben wird.	
Wärmeübergangskoeffizient α	der Wärmeübergangskoeffizient an der Heizkörperoberfläche	
Raumtemperatur ϑ	die aktuelle Raumtemperatur	

Die Funktionen des Heizgerätetyps II mit Regelung unterscheiden sich namentlich nicht von denen des Heizgerätetyps I mit Regelung.

Funktion	*Parameter*	*Bemerkung*
inputInformation	Power ON/OFF	
inputInformation	Raumtemperatur	
outputEnergie		
wärmefluss	Raumtemperatur	

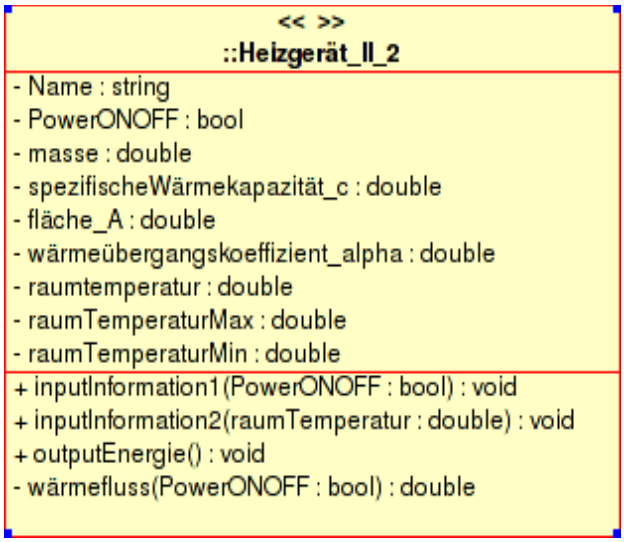


Abb. 14 Darstellung des Heizgerätetyps II mit Regelung in UML

7.3 Transformation des abstrakten Modells des Technischen Systems *Haus* von UML in die Programmiersprache Java

Dieses Kapitel zeigt nun, wie sich auf der Basis der systematischen Vorarbeit ein Simulationsmodell erstellen lässt. Einige zusätzliche Objekte sind erforderlich, damit aus dem statischen Modell ein dynamisches werden kann. Abb.15 zeigt noch einmal den angestrebten Aufbau der Konstruktions- und Simulations-Umgebung „**HausSim**“.

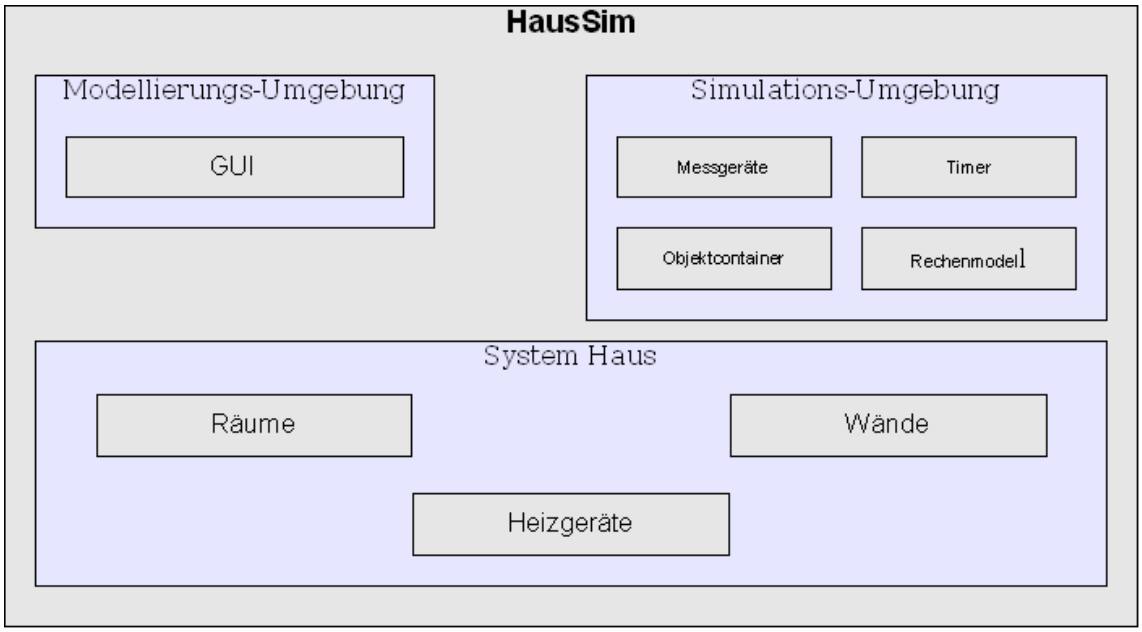


Abb. 15 Objekthierarchie der SiMo

Die SiMo-Umgebung **HausSim** besteht aus drei großen Systemen:

1. Aus dem System **Haus**, in dem die bekannten Subsysteme gemäß den vorherigen Überlegungen zusammengefasst sind.

2. Aus dem System **Modellierungs-Umgebung,** das alle Objekte beinhaltet, die notwendig sind, um die anderen Objekte aus dem System **Haus** und dem System **Simulations-Umgebung** zu konfigurieren. Die Modellierungsumgebung beinhaltet die General User Interfaces (GUIs). GUIs sind die Benutzerschnittstellen, mit deren Hilfe der User die Objekte grafisch konfigurieren kann.

   In Abb. 16 ist ein Beispiel für ein GUI dargestellt. Es handelt sich um das GUI für die Konfiguration der Raumobjekte der hier zu erstellenden SiMo-Umgebung.

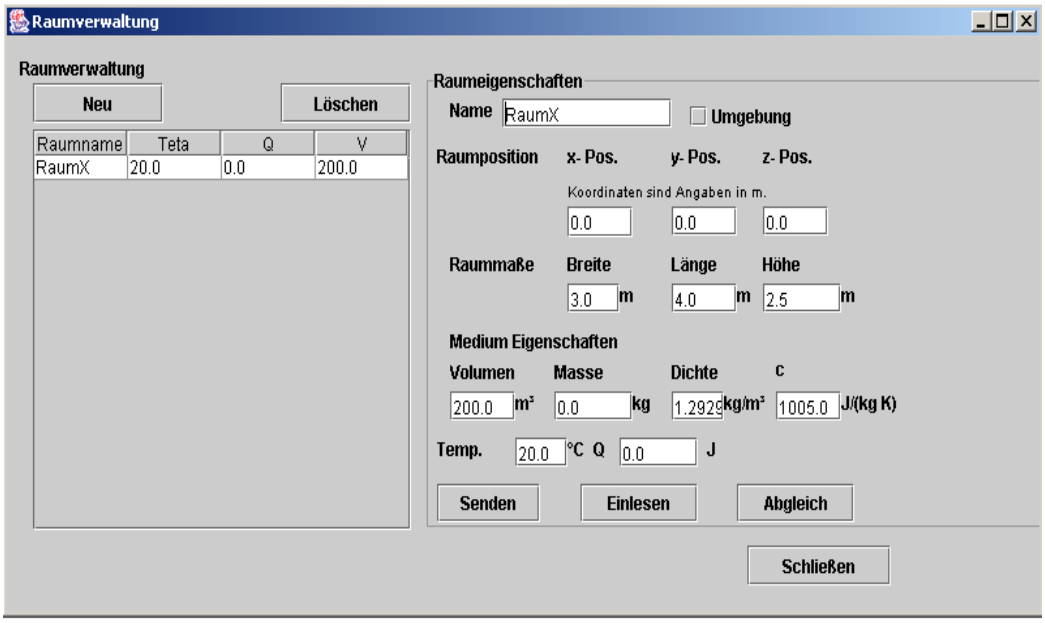


Abb. 16 Beispiel eines GUI (Raumkonfiguration der SiMo-Umgebung)

3. Aus dem System **Simulations-Umgebung**, in dem alle anderen Objekte gesammelt werden, die für die Organisation und den dynamischen Ablauf der Simulation verantwortlich sind. Das System **Messgeräte** verwaltet alle virtuellen Messgeräte, die während der Simulation eingesetzt werden. Der **Timer** ist nötig, um die Objekte zeitlich zu synchronisieren. Im **Objektcontainer** werden alle Objekte der Simulation verwaltet. Das **Rechenmodell** sorgt dafür,

   dass erstens alle Objekte zum richtigen Zeitpunkt aktiviert werden die Berechnungen durchzuführen,

dass zweitens die neuen Werte an die GUIs weitergegeben werden, und dass drittens die Messgeräte ausreichend Zeit zur Verfügung haben, die neuen Daten darzustellen.

Der Aufbau und die Funktionsweisen der einzelnen Objekte werden in den folgenden Kapiteln noch näher erläutert werden.

Zunächst einmal wird das mathematische Grundgerüst der Simulation erläutert, damit die Hausobjekte mit ihren entsprechenden Funktionen richtig erstellt werden können.

7.3.1 Das mathematische Modell der Simulation

Auswahl der Berechnungsmethode

Da es sich in diesem Fall um ein ganzheitliches Problem handelt, können die einzelnen Aktionen der Subsysteme nicht getrennt voneinander betrachtet und berechnet werden, da jede kleinste Änderung in einem Subsystem sofort eine Reaktion in den übrigen Subsystemen bewirkt. Somit müssen nach jeder Änderung eines beliebigen Parameters alle Subsysteme neu berechnet werden. Das bedeutet, dass auch bei jedem Zeitpunktwechsel eine Neuberechnung durchgeführt werden muss. Da die Simulation es zulässt, beliebige Änderungen am System während der Laufzeit vorzunehmen, kommen nur wenige mathematische Berechnungsverfahren in Betracht.

Euler

Der Autor hat sich für das einfache Eulersche Näherungsverfahren auf Grundlage der Differenzengleichungen entschieden, da trotz der Ungenauigkeiten, die bei dieser Methode auftreten, die Vorteile überwiegen .

1. Es kann mit einfachen statischen Formeln gerechnet werden.

   Da das hier vorgestellte Konzept dafür ausgelegt ist, technisches Denken und Handeln einer heterogenen Zielgruppe[8] zu fördern und nicht für den Ge-

8 in Bezug auf mathematische Fähigkeiten

brauch durch einige wenige Experten[9], werden wenig komplexe mathematische Berechnungen und physikalische Formeln verwendet, die der Lerner aus jeder Formelsammlung entnehmen kann. Er muss lediglich wissen, dass die Formeln statische Vorgänge beschreiben, und dass dadurch ein Fehler entsteht, der in erster Linie von den ausgewählten Zeitintervallen abhängt.

2. Die Berechnung kann so gestaltet werden, dass zu jedem Zeitpunkt sämtliche Randbedingungen neu definiert und berücksichtigt werden können. Somit kann eine gewisse Unabhängigkeit der Subsysteme erreicht werden.
3. Die schnellen Rechengeschwindigkeiten moderner Computer erlauben das Zeitintervall Δt in den Berechnungen entsprechend klein zu wählen, um für den angestrebten Zweck angemessene Ergebnisse zu erzielen.
4. Gesetzmäßigkeiten können in den Objekten abgelegt werden, womit eine Kapselung der Objekte erreicht werden kann.

Finite Elemente-Methode

Mit der **FEM** (Finite Elemente- Methode) können sehr komplexe und genaue Bilder der Wärmeverteilung im Gebäude erstellt werden. Diese Methode ist jedoch für die Ziele dieses Konzeptes ungeeignet.

Einige der Gründe sind:

1. Die Programmierung der Methode setzt ein fundiertes und erweitertes mathematisches Grundwissen voraus.

2. Selbst die Anwendung der FEM mit speziellen Modellierungs-Tools erfordert vom User ein fundiertes Wissen über die genauen Sachzusammenhänge seines Modells und die mathematischen Zusammenhänge. Selbst bei der Erstellung der Berechnungsnetze mit guten FEM-Tools können kleine aber folgenschwere Fehler gemacht werden, die absolut unbrauchbare Ergebnisse liefern.

9 Die Experten haben das Ziel, schnelle und optimale Ergebnisse zu bekommen. Für sie ist Genauigkeit und Schnelligkeit oberstes Gebot. Dabei werden Berechnungen soweit verfeinert, dass sie für Einsteiger zu komplex und nicht mehr nachvollziehbar werden.

3. Die richtige Erstellung des Berechnungsnetzes setzt sehr viel Erfahrung voraus.

Gleichungssysteme

Die Berechnungen über Gleichungssysteme zu lösen, wie es z. B. in der Elektrotechnik bei der Berechnung von Netzwerken häufig praktiziert wird, wurde zu Anfang auch in Betracht gezogen, dann jedoch wieder verworfen Der Grund war, dass es wieder Einschränkungen bezüglich der flexiblen Ausbaufähigkeit des gesamten Modells gegeben hätte. Das Grundprinzip der Objektorientierung und der Systembildung, das in diesem Konzept umgesetzt ist, lässt sich mit diesem Ansatz nicht schlüssig ausführen. Für jedes neu hinzugefügte Subsystem oder Parameter müsste das Gleichungssystem modifiziert werden, was wiederum ein erweitertes mathematisches Grundwissen und erweiterte Kenntnisse in der Programmierung voraussetzt. Die Stärke des Konzepts soll in seiner Einfachheit liegen. Optimierungen können jederzeit im Nachhinein vorgenommen werden.

Die Objekte stellen die Gleichungen zur Verfügung. Diese Gleichungen müssen keiner Konvention unterliegen, die es ermöglichen würde, diese in ein Gleichungssystem einzubinden, um dann über die üblichen Lösungsverfahren für Gleichungssysteme (z. B. Gauß-Algorithmus) gelöst zu werden.

Mit Hilfe des Modells der Objekte kann man zwar die statischen Abhängigkeiten und Funktionalitäten darstellen, zu einer Simulation fehlt jedoch noch ein Rechenmodell, um dynamische Prozesse darstellen zu können.

Um möglichst viele Zielgruppen zu ereichen, soll hier ein relativ einfach nachvollziehbares Rechenmodell verwendet werden. Den Schülern/innen können auf diese Weise die Grundlagen der Differenzialrechnung anschaulich erklärt werden. Bei Studenten könnte noch zusätzlich die praktische Umsetzung von Folgen und Reihen thematisiert werden.

Die besondere Stärke des Rechenmodells liegt jedoch darin, dass es auch ohne die Einbeziehung von Differenzialrechnung, Folgen und Reihen erklärt werden kann.

Das Rechenmodell betrachtet das System **Haus** abstrakt als aus Subsystemen bestehend, die Energiepotenziale (Wärmepotenziale) darstellen und weiteren Subsystemen, die Energieflüsse (Wärmeflüsse) zulassen. Die Räume, inklusive der Umgebung, sind die Wärmepotenziale, die versuchen sich ständig durch die Wände auszugleichen.

Physikalische Grundlagen

Das Wärmepotenzial (Wärmeinhalt des Raumes) wird mit Formel (1) berechnet. Das Nullpotenzial wird auf 0°C festgelegt.

$$Q = mc\vartheta \qquad (1)$$

Der Fluss durch die Wand wird mit Formel (2) errechnet. In Formel (3) ist die Formel so aufgelöst, wie sie hier verwendet wird.

$$\Phi = \frac{\Delta Q}{\Delta t} = kA(\Delta\vartheta) \qquad (2)$$

$$\Phi = \frac{\Delta Q}{\Delta t} = kA(\vartheta_{Raum1} - \vartheta_{Raum2}) \qquad (3)$$

Anhand von Formel (3) wird ersichtlich, dass zum Berechnen des Wärmeflusses die Raumtemperaturen benötigt werden. Da aber während der Rechenabfolge der Simulation die Wärmeinhalte aufsummiert werden, muss Formel (1) nach ϑ umgestellt werden (s. Formel (4)).

$$\vartheta = \frac{Q}{mc} \qquad (4)$$

Was passiert jetzt zeitlich gesehen im Haus?

Die Potenziale versuchen sich auszugleichen. Dabei hängt der Wärmefluss von der Temperaturdifferenz in den anliegenden Räumen ab, d. h. hier liegt eine Abhängigkeit vor, die eine einfache Berechnung unmöglich erscheinen lässt. Wird hier berücksichtigt, dass bei dem Hausmodell nicht nur zwei Räume mit einer Wand betrachtet werden sollen, sondern beliebig viele Räume und Wände, wäre die Erstelung und Lösung eines Gleichungssystem erforderlich, um die Wärmepotenziale und Wärmeflüsse für jeden Zeitpunkt zu erhalten. Angesichts der Tatsache, dass dieses Programm auch für Schüler geeignet sein soll, wäre es eine zu komplexe Vorgehensweise.

Die Lösung liegt in der Rechengeschwindigkeit des Computers. So kann ein numerisches Näherungsverfahren verwendet werden, das das Lösen von Gleichungssystemen nicht erfordert und zudem noch den Vorteil hat, dass beliebige Parameter der Objekte während der Berechnungen geändert werden können.

Der konkrete Berechnungsablauf der Simulation

Die Temperatur und die Abmessungendes Raumes sind Größen, die dem Lerner bekannt sind und die er bestimmen kann. Die Masse des Mediums im Raum kann er nicht direkt angeben, jedoch das Volumen über die Raummaße bestimmen. Die spezifische Wärmekapazität und die Dichte des Mediums im Raum können vorgegeben werden, oder sind in Tabellen nachzuschlagen.

Anhand dieser Vorgaben muss das Programm die Räume und die Wände zu Beginn der Simulation erst initialisieren. Dies geschieht durch den folgenden Berechnungsablauf.

Initialisieren der Objekte:

Räume:

1.Volumina der Räume berechnen:

$$V = H\ B\ T \tag{5}$$

2. Masse des Mediums in den Räumen berechnen:

$$m = \frac{\rho}{V} \tag{6}$$

3. Wärmeinhalte der Räume berechnen:

$$Q = mc\vartheta \tag{7}$$

Wände:

4. Flächen der Wände berechnen:

$$A = H\ B \tag{8}$$

Timer initialisieren:

5. Δt [10] des Timers auf 1 Sekunde setzen:

$$\Delta t = 1 \tag{9}$$

Berechnungen der Simulation beginnen:

6. Temperaturen der Räume bestimmen:

$$\vartheta = \frac{Q}{mc} \tag{10}$$

10 Das Zeitintervall des Timers.

7. Wärmemenge, die jeweils durch die Wände fließt, berechnen:

$$\Delta Q = kA\,dt\,(\vartheta_{Raum1} - \vartheta_{Raum2}) \qquad (11)$$

8. Wärmemenge aus (10) den Räumen anrechnen:

$$Q_n = Q_{n-1} + \Delta t \qquad (12)$$

9. Springe zu Punkt 6[11]

Die Wahl von Δt beeinflusst die Genauigkeit der Berechnungen. Wichtig ist, dass zunächst alle Potenziale berechnet werden und anschließend die Flüsse. Mit Hilfe dieses Algorithmus können durch sehr kleine Δt sehr hohe Rechengenauigkeiten erreicht werden. Ein Nachteil ist jedoch, dass der Ablauf der Systemzeit gedehnt wird. Da es sich jedoch um eine Lernsoftware handeln soll und nicht um eine Anwendung für professionelle Gebäudeberechnungen, können bei großem Δt auch entsprechende Ungenauigkeiten in den Berechnungen toleriert werden.

Fehlerbetrachtung für das Näherungsverfahren

Bei dem hier verwendeten Näherungsverfahren werden durch die Wahl von eines zu großen Δt gleich mehrere Fehler verstärkt. Diese treten bei der Berechnung der Wärmemengen auf, die von einem Wärmepotenzial (Raum oder Heizgerät) in ein anderes fließen. Sie treten alle gleichzeitig auf und hängen teilweise voneinander ab. Es kann von einem Fehlerquotienten für

$$F_{[\Delta Q]} = \frac{\Delta Q_{\Delta t_1}}{\Delta Q_{\Delta t_2}} \qquad (13)$$

gesprochen werden. Wobei $\Delta Q_{\Delta t_n}$ die Wärmemenge für ein bestimmtes Δt ist.

$$\Delta Q_{\Delta_n} = UA \cdot (\vartheta_{n1} - \vartheta_{n2}) \cdot \Delta t_n \qquad (14)$$

11 Es müssen die Wärmeinhalte der Räume neu berechnet werden, da der User zwischendurch die Raumeigenschaften neu belegen kann.

Anhand von Formel (13) lassen sich die ersten Quellen der Ungenauigkeiten erkennen. Der Faktor UA , bestehend aus dem Wärmedurchgangskoeffizienten U und der Wandfläche A, lässt den Fehler proportional ansteigen. Das heißt, je größer U oder A bei sonst konstanten Bedingungen wird, desto größer wird auch die Ungenauigkeit.

Mit dem nächsten Faktor $\Delta \vartheta_n$ ergibt sich eine weitere Fehlerquelle, die gleich mehrere Ursachen hat.

Erstens wird dieser Faktor im Laufe der Berechnungen exponentiell kleiner, da sich die Temperaturen annähren. D. h. der Fehler wird im Laufe des Berechnungszyklus kleiner.

Zweitens sind die Werte der einzelnen Raumwärmepotenziale, aus denen ϑ_n berechnet wird, fehlerbehaftet. Diese Fehler beruhen einmal auf der Ungenauigkeit der einzelnen ΔQ , die während eines bestimmten Δt in den Raum hinein- und herausfließen. Weiterhin werden diese fehlerbehafteten ΔQ durch die zwangsläufige Summation zu einem bestimmten Zeitpunkt aufsummiert.

Da das System **Haus** vom User durch eine Vielzahl von Parametern bestimmt und variert werden kann und aufgrund der oben erläuterten Fehlerquellen, kann an dieser Stelle keine konkrete Empfehlung für geeignete Δt -Werte ausgesprochen werden. Als Anhaltspunkt kann gelten, dass bei großen Temperaturunterschieden, großen Wandflächen, geringer Dämmung und einer großen Anzahl an Wänden das Δt klein gewählt werden muss, um genaue Berechnungen zu ermöglichen. Ein geeigneter Wert muss durch gezieltes Ausprobieren gefunden werden. Da es sich bei der SiMo-Umgebung in erster Linie um eine Lernsoftware handelt, ist die Frage der Genauigkeit nicht vorrangig, da die wesentlichen Abhängigkeiten und Kurvenverläufe der Vorgänge im System **Haus** auch mit großen Fehlertoleranzen erkennbar sind.

7.3.2 Die Umsetzung des theoretischen Systemmodells in Java

In diesem Kapitel wird die Erstellung der Objekte erklärt, die sich später einfach in die Simulation integrieren lassen. Es hängt von der richtigen Planung zu Beginn der Programmierung ab, ob sich die Simulation im Nachhinein einfach und logisch erweitern lässt. Die objektorientierte Programmierung erlaubt, dieselben Überlegungen, wie sie bei der Systemtheorie erfolgt sind, auch hier anzuwenden und zu konkretisieren.

Das System wird zum Systemobjekt

In Kapitel 7.1.3 wurde ein System unter dem funktionalen Aspekt mit Eigenschaften (Zustände) und Input- und Output-Werten charakterisiert. In Abb. 17 wird ein solches Objekt in UML dargestellt.

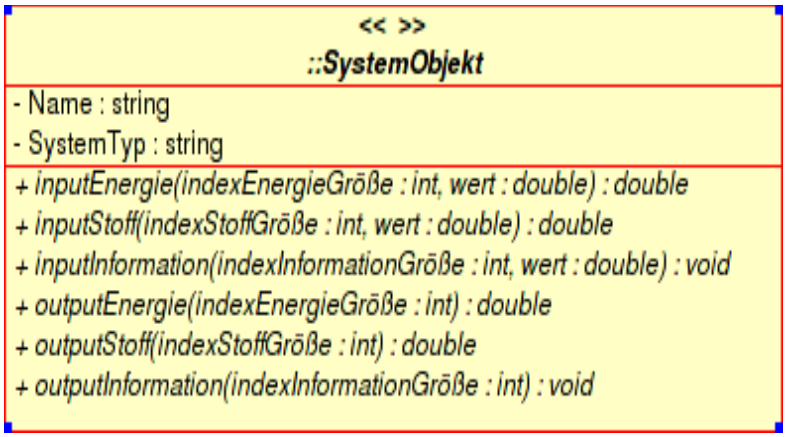


Abb. 17 Systemobjekt in UML-Darstellung

Das System bekommt als Eigenschaft einen **Namen** zur Identifikation. Die weitere Eigenschaft **SystemTyp** dient als Hinweis, um welches Subsystem oder Objekt es sich hier handelt. Diese Eigenschaft kann erst belegt werden, wenn das abstrakte Objekt durch weitere Beschreibungen konkretisiert wurde, oder wenn ein anderes Objekt darauf aufbaut. Es handelt sich bei dem **SystemOb-**

jekt um eine abstrakte Klasse, weil hier nur beschrieben wird, welche Funktionalität Objekte aufweisen müssen, damit sie als **SystemObjekt** akzeptiert werden.

Das im Programm verwendete **SystemObjekt** in UML sieht wie folgt aus (Abb. 18).

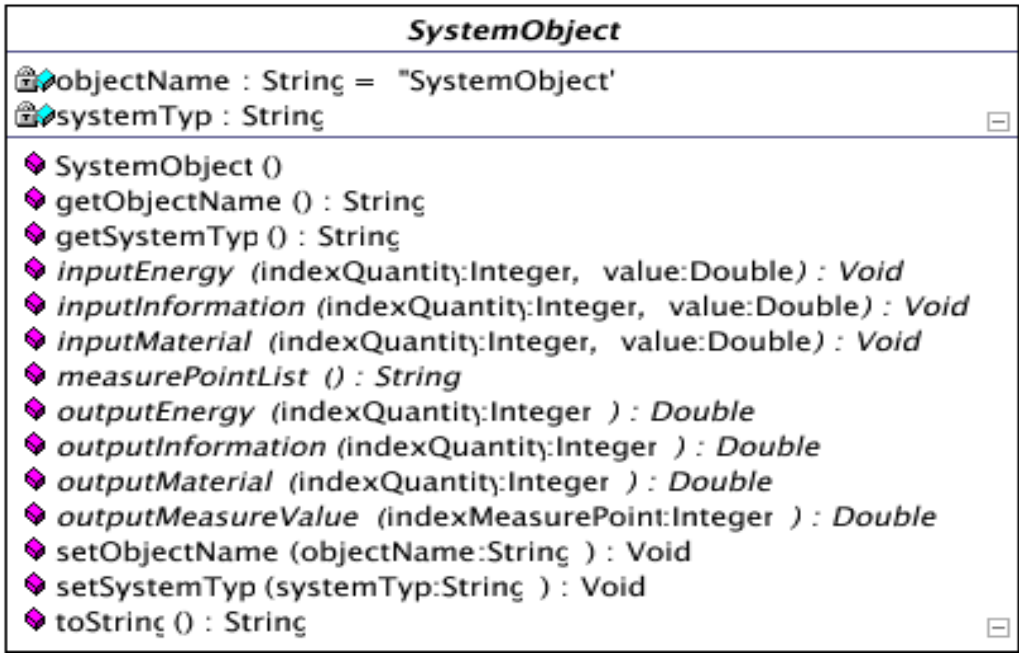


Abb. 18 Darstellung des Systemobjekts in UML

Im Java-Programmcode wird schnell ersichtlich, dass alle Eigenschaften und Funktionen, die in UML übersichtlich untereinander und nach Eigenschaften und Methoden aufgelistet sind, auch in Java sortiert untereinander stehen (siehe blau unterlegter Code).

```
package houseObjects;

public abstract class SystemObject implements interfacesHou-
se.MeasuredObject
{
    /** Name des Systemobjekts**/
    private String objectName = "SystemObject";

    /** Angabe zum HausSystemtyp**/
    private String systemTyp;

    public SystemObject()
```

```
    {
    }
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract double outputMeasureValue(int indexMeasu-
rePoint);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract String measurePointList();
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract void inputEnergy(int indexQuantity, dou-
ble value);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract void inputMaterial(int indexQuantity,
double value);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract void inputInformation(int indexQuantity,
double value);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract double outputEnergy(int indexQuantity);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract double outputMaterial(int indexQuantity);
    /** Diese Funktion wird von den Hausobjekten Raum, Wand
und Heizkörper implementiert**/
    public abstract double outputInformation(int indexQuanti-
ty);

    /**Holt den Objekt-Namen
     * @return Objektname.
     *
     */
    public String getObjectName()
    {
        return this.objectName;

    }
    /** Setzt den Objekt-Namen
     * @param objectName Neuer Wert des Objekt-Namen.
     *
     */
    public void setObjectName(String objectName)
    {
        this.objectName = objectName;

    }
    /** Holt den System-Typ
     * @return System-Typ.
     *
     */
    public String getSystemTyp()
    {
        return this.systemTyp;
    }
    /** Setzt den System-Typ
     * @param systemTyp Neuer Wert des Objekt-Namen
     *
     */
    public void setSystemTyp(String systemTyp)
```

```
    {
        this.systemTyp = systemTyp;
    }

    /** Bestimmt den String, der das systemobjekt darstellen
soll**/
    public String toString()
    {
        return this.objectName;
    }


}
```

Programm Beisp.1: Programmcode der Klasse Systemobjekte; die abstrakten Input- und Outputmethoden sind blau markiert

Ohne im Detail auf die genaue Java-Programmierung einzugehen, sollen einige Hinweise gegeben werden, damit der Leser im Programmcode sich zurechtfinden kann.

Die Zeile <public abstract class SystemObject implements interfacesHouse.MeasuredObject > ist der Kopf der Klasse. Danach folgen die Eigenschaften, anschließend die abstrakten Funktionen. Hinter den abstrakten Funktionen gibt es zwei Getter- und Setter-Funktionen, die dazu dienen, die jeweiligen Eigenschaften zu setzen und zu lesen. In diesem Fall sind es die Eigenschaften **objectName** und **systemTyp**. Diese Funktionen werden immer benötigt, um später auf die Eigenschaften zugreifen zu können. Die GUIs sind ein gutes Beispiel für Objekte, die mit Hilfe dieser Funktionen die Werte der Objekte lesen und ändern.

Diese Klasse dient nur zur Vereinfachung und Verallgemeinerung der Systemobjekte und wird automatisch von den Hausobjekten (Raum, Wand und Heizgerät) übernommen.
Ein weitere Vereinfachung und Verallgemeinerung stellt das Objekt **HausObjekt** dar.

Die Klasse HausObjekte

Um die Möglichkeit zu wahren in einer späteren Version der Simulationsumgebung die Anordnung von Räumen zu visualisieren, wird eine spezielle Superklasse der normalen Haus-Objekte benötigt. Diese Klasse wird hier **HausObjekte** genannt. Die Klasse **HausObjekte** ist wiederum eine abstrakte Klasse,

die jedoch in einigen Eigenschaften etwas konkreter wird. Sie beinhaltet für alle Objekte, die unter die Klasse **HausObjekte** fallen, die Position im Objekt **Haus**. Mit **Position** sind die Koordinaten im Haus gemeint. Aus diesen Koordinaten kann später z. B. ein Grundriss generiert werden. Zusätzlich wird den Objekten ein Objekt **Rechteck** zugefügt, das zur Darstellung eines Raumes, einer Wand oder eines Heizgerätes ausreichen soll. Die Maße dieses Rechtecks werden durch die Eigenschaften des konkreten Objektes festgelegt. Die weitere Eigenschaft **Ausrichtung** legt die Ausrichtung des Objektes fest, z. B. für die Wand. (Bei der Wand gibt es nur die Eigenschaften Höhe und Breite. Somit ist in einer X-Y-Darstellung nicht eindeutig festgelegt, ob die Breite der Wand in X-Richtung oder in Y-Richtung gemeint ist.)

Das Systemobjekt **HausObjekt** sieht dann in UML aus, wie in Abb. 19 dargestellt.

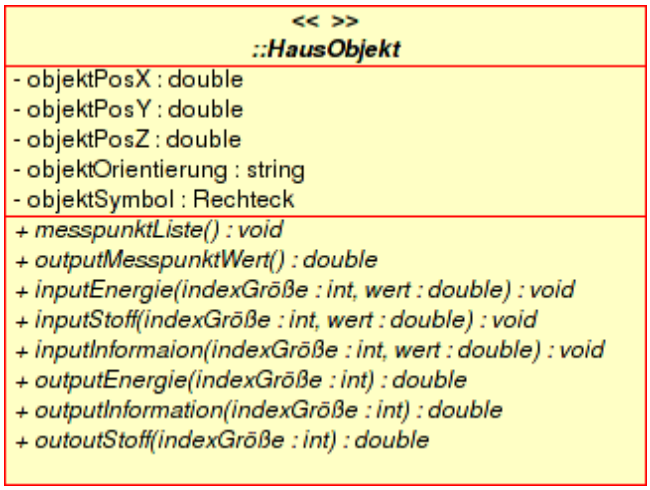


Abb. 19 HausObjekt in UML-Darstellung

In dieser Darstellung wird ersichtlich, dass noch einmal die abstrakten Funktionen der Inputs und Outputs aufgeführt werden, aber noch nicht konkretisiert werden. Die bisher noch abstrakten Funktionen werden im nächsten Schritt konkretisiert.

Von der Objektklasse **HausObjekt** leiten sich nun die drei in der Simulation verwendeten Systemobjekte ab. Abb. 20 zeigt die Vererbungshierarchie.

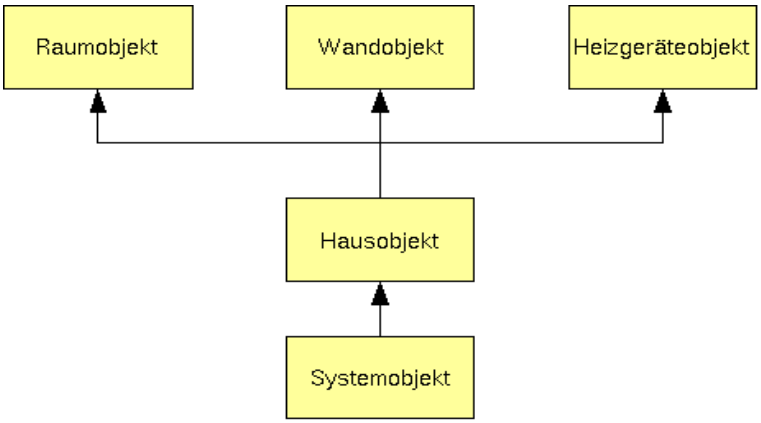


Abb. 20 Vererbungshierarchie

Das Haus-Objekt Raum.

Die Objekte der Klasse **Räume** sind die ersten Objekte, die konkretisiert und erstellbar sind. Sie erben die konkreten Eigenschaften und Methoden der Superklassen **HausObjekte** und **Systemobjekte** und konkretisieren die abstrakten Funktionen der beiden Superklassen. Die konkrete Klasse **RoomClass**, wie sie in der Simulation verwendet wird, wird in Abbildung Abb. 21 in UML dargestellt.

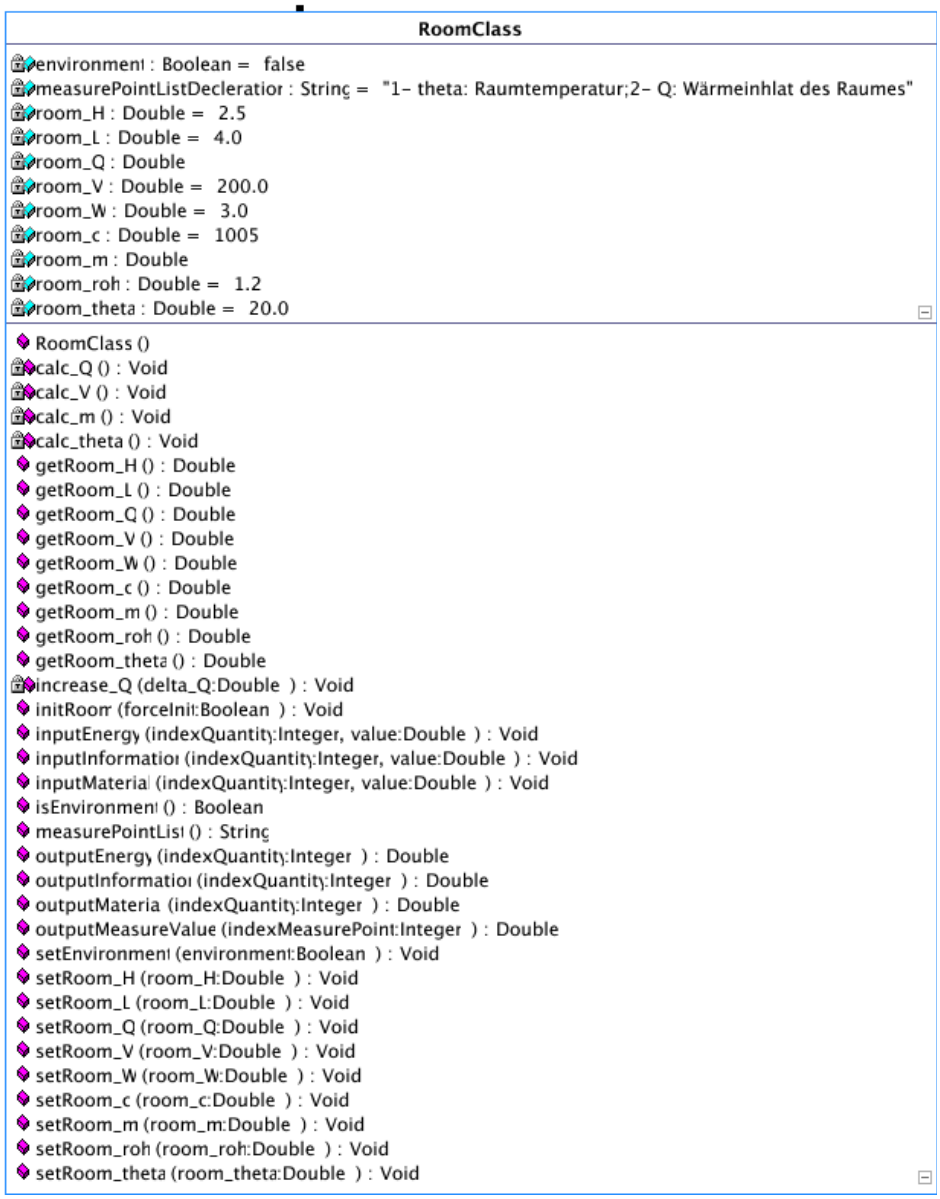


Abb. 21 UML-Darstellung des verwendeten Raumobjektes in der Simulation

Die Klasse RoomClass (s. Abb. 21) weist die Eigenschaft **environment** und die Methoden **init_Room()**, **clac_Q()**, **clac_V(), calc_m()** und **calc_theta()** auf.

Die Eigenschaft **environment** ist notwendig, damit das Programm unterscheiden kann, ob es sich bei dem Raum-Objekt um einen Raum im Haus (be-

grenztes Volumen) oder um die Umgebung (unendliches Volumen) handelt. Diese Unterscheidung ist insbesondere für die Methode **calc_theta()** wichtig (siehe Funktion **calc_theta()**)

init_Room(): Die Methode **init_Room()** initialisiert den Raum. Sie berechnet aus den anderen Eigenschaften die Werte für die Eigenschaften V, m und Q. Der Users kann eine Initialisierung unterbinden, wenn die eingetragenen Werte für V oder m nicht aus den Raumeigenschaften berechnet werden sollen.

Die mit **calc**_ beginnenden Methoden, beinhalten die Berechnung der Größe , die in den Eigenschaften festgelegt wird. Dabei bietet die Methode **calc_theta()** eine Besonderheit. Sie unterscheidet zwischen dem Raum im Haus und dem Raum Umgebung (environment). Im Falle eines Raum-Objekts **Raum** im Haus(Eigenschaft: environment = false) wird die Temperatur aus dem Wärmeinhalt Q des Raumes berechnet, sonst bleibt die Temperatur konstant.

Die Methode **input_Energie()** fügt dem Raum eine bestimmte Wämemenge zu, indem seine Eigenschaft Q um den übergebenen Wert erhöht oder reduziert wird. **output_Energie()** hingegen berechnet die Temperatur des Raumes aus der Eigenschaft des Raumes Q neu und stellt auf diese Weise den anderen Objekten das ganze Wärmepotenzial des Raumes zur Verfügung. Die genauen Interaktionen der Objekte untereinander werden in Kapitel 7.3.5 beschrieben.

Obwohl der Raum kein informationsumsetzendes System ist, muss aus programmiertechnischer Notwendigkeit ebenfalls die Funktion **outputInformation()** für den Raum belegt werden. Hier wird die Temperatur als Information an das anfragende Objekt weitergegeben (s. Kapitel 7.3.5).

Die Methoden **outputMeasureValue()** und **measurePointList()** dienen dazu, dass Messgeräte auf eine einfache Weise Messpunkte im Raumobjekt setzen können (s. Kapitel 7.3.5).

Nach der Transformation des Raumobjektes in die Programmiersprache Java wird erkennbar, dass sich die Denkweise und die Darstellung des ehemals theoretischen Subsystems **Raum** und der jetzt konkretisierten Klasse **RoomClass** nicht wesentlich geändert haben. Im letzten Schritt wurde lediglich der Raum mit den mathematischen Gesetzmäßigkeiten beschrieben. Durch die Verfeinerung des mathematischen Modells des Raums lassen sich jetzt beliebige Eigenarten des Raums im Nachhinein hinzufügen.

Im folgenden Programmcodebeispiel kann gezeigt werden, wie einfach die Methoden definiert werden, so dass auch nicht-professionelle Programmierer schnell einen Zugang zur Klasse **RoomClass** erhalten können.

```
package houseObjects;
import nature.NatureConstants;
public class RoomClass extends HouseObjectClass
{

    /** Holds value of property room_Q. Wärmeinhalt des Raum-
es */
    private double room_Q;

    /** Holds value of property room_theta. Temperatur imple-
ments Raum*/
    private double room_theta = 20.0;

    /** Holds value of property room_m. Masse des Mediums im
Raum (Luft)*/
    private double room_m;

    /** Holds value of property room_c. spez. Wärmekapazität
des Medium (Luft) in (J/kg K) */
    private double room_c = 1005;

    /** Holds value of property room_V. Raumvolmen */
    private double room_V= 200.0;

    /** Holds value of property room_W. Width- Breite des
Raumes x-Achse */
    private double room_W= 3.0 ;

    /** Holds value of property room_L. Length - Länge des
Raumes y-Achse */
    private double room_L= 4.0;

    /** Holds value of property room_H. Hight - Höhe des
Raumes z-Achse*/
    private double room_H= 2.5;

    /** Holds value of property room_roh. Dichte des Mediums
(Luft) */
```

```
    private double room_roh= 1.2;

    /** Holds value of property environment. */
    private boolean environment = false;

    private String measurePointListDecleration = "1- theta:
Raumtemperatur;2- Q: Wärmeinhlat des Raumes";

    /** Creates a new instance of Räume */
    public RoomClass()
    {
        this.setObjectName("RaumX");
        this.setSystemTyp("Raum");
        this.room_c = nature.NatureConstants.air_c_20();
        this.room_roh = nature.NatureConstants.air_roh_20();

    }

    /** Getter for property room_Q.
     * @return Value of property room_Q.
     *
     */
    public double getRoom_Q()
    {
        return this.room_Q;
    }

    /** Setter for property room_Q.
     * @param room_Q New value of property room_Q.
     *
     */
    public void setRoom_Q(double room_Q)
    {
        this.room_Q = room_Q;
    }

    /** Getter for property room_theta.
     * @return Value of property room_theta.
     *
     */
    public double getRoom_theta()
    {
        return this.room_theta;
    }

    /** Setter for property room_theta.
     * @param room_theta New value of property room_theta.
     *
     */
    public void setRoom_theta(double room_theta)
    {
        this.room_theta = room_theta;
    }

    /** Getter for property room_m.
     * @return Value of property room_m.
     *
     */
    public double getRoom_m()
    {
        return this.room_m;
    }
```

```
    /** Setter for property room_m.
     * @param room_m New value of property room_m.
     *
     */
    public void setRoom_m(double room_m)
    {
        this.room_m = room_m;
    }

    /** Getter for property room_c.
     * @return Value of property room_c.
     *
     */
    public double getRoom_c()
    {
        return this.room_c;
    }

    /** Setter for property room_c.
     * @param room_c New value of property room_c.
     *
     */
    public void setRoom_c(double room_c)
    {
        this.room_c = room_c;
    }

    /** Getter for property room_V.
     * @return Value of property room_V.
     *
     */
    public double getRoom_V()
    {
        return this.room_V;
    }

    /** Setter for property room_V.
     * @param room_V New value of property room_V.
     *
     */
    public void setRoom_V(double room_V)
    {
        this.room_V = room_V;
    }

    /** Getter for property room_w.
     * @return Value of property room_w.
     *
     */
    public double getRoom_W()
    {
        return this.room_W;
    }

    /** Setter for property room_w.
     * @param room_w New value of property room_w.
     *
     */
    public void setRoom_W(double room_W)
    {
        this.room_W = room_W;
    }
```

```
    /** Getter for property room_l.
     * @return Value of property room_l.
     *
     */
    public double getRoom_L()
    {
        return this.room_L;
    }

    /** Setter for property room_l.
     * @param room_l New value of property room_l.
     *
     */
    public void setRoom_L(double room_L)
    {
        this.room_L = room_L;
    }

    /** Getter for property room_H.
     * @return Value of property room_H.
     *
     */
    public double getRoom_H()
    {
        return this.room_H;
    }

    /** Setter for property room_H.
     * @param room_H New value of property room_H.
     *
     */
    public void setRoom_H(double room_H)
    {
        this.room_H = room_H;
    }
    /** Berechnet den Wärmeinhalt des Raumes */
    private void calc_Q()
    {
        this.room_Q= this.room_m*this.room_c*this.room_theta;
    }
    /** Berechnet die Temperatur des Raumes */
    private void calc_theta()
    {   if (this.environment)
        {
            this.room_theta = this.room_theta;
        }
        else
        {
            this.room_theta=this.room_Q/(this.room_c*this.roo
m_m);
        }
    }
    /** Berechnet die Masse des Mediums (Luft) im Raum */
    private void calc_m()
    {
        this.room_m=this.room_roh*this.room_V;
    }

    /** Getter for property room_roh.
     * @return Value of property room_roh.
     *
     */
    public double getRoom_roh()
```

```
    {
        return this.room_roh;
    }

    /** Setter for property room_roh.
     * @param room_roh New value of property room_roh.
     *
     */
    public void setRoom_roh(double room_roh)
    {
        this.room_roh = room_roh;
    }
    /** Berechnet das Volumen des Raumes */
    private void calc_V()
    {
        this.room_V=this.room_L*this.room_W*this.room_H;
    }

    private void increase_Q(double delta_Q)
    {
        this.room_Q= this.room_Q + delta_Q;
    }

    /** Initilalisiert den Raum, damit alle Größen schlüssig
sind, wenn eine Größe Null ist.*/
    public void initRoom(boolean forceInit)
    {
        if (forceInit)
        {
            this.calc_V();
            this.calc_m();
            this.calc_Q();
        }
        else
        {
            if (this.room_V == 0)
            {
                this.calc_V();
            }
            if (this.room_m == 0)
            {
                this.calc_m();
            }
            if (this.room_Q == 0)
            {
                this.calc_Q();
            }
        }
    }

    /** Getter for property environment.
     * @return Value of property environment.
     *
     */
    public boolean isEnvironment()
    {
        return this.environment;
    }

    /** Setter for property environment.
     * @param environment New value of property environment.
     *
     */
```

```
    public void setEnvironment(boolean environment)
    {
        this.environment = environment;
    }
    /** Gitbt die Messpunkte des Raumobjectes zurück**/
    public String measurePointList()
    {
        return this.measurePointListDecleration;
    }

    public double outputMeasureValue(int indexMeasurePoint)
    {
        double measureOutput = 0;
        switch (indexMeasurePoint)
        {
            case 1:
                measureOutput = this.room_theta;
                break;
            case 2:
                measureOutput = this.room_Q;
                break;
        }
        return measureOutput;

    }
    public void inputEnergy(int indexQuantity, double value)
    {
        switch (indexQuantity)
        {
            case 1:
                this.increase_Q(value);
                break;
        }

    }

    public void inputInformation(int indexQuantity, double
value)
    {
        switch (indexQuantity)
        {
            case 1:
                break;
        }
    }

    public void inputMaterial(int indexQuantity, double va-
lue)
    {
        switch (indexQuantity)
        {
            case 1:
                break;
        }
    }
    /** Berechnet nur die Temperatur des Raumes aus Q, diese
wird als Wärmepotential am Ausgang bereitgestellt.**/
    public double outputEnergy(int indexQuantity)
    {
        double value =0;
        switch (indexQuantity)
        {
```

```
            case 1:
                this.calc_theta();
                value =this.room_theta;
                break;
        }
        return value;
    }

    public double outputMaterial(int indexQuantity)
    {
        double value = 0;
        switch (indexQuantity)
        {
            case 1:
                value =0;
                break;
        }
        return value;
    }
    public double outputInformation(int indexQuantity)
    {
        double value = 0;
        switch (indexQuantity)
        {
            case 1:                          /*Gibt die Raum-
temperatur raus*/
                value = this.room_theta;
                break;
        }
        return value;
    }
}
```

Programm Beisp. 2: Programmcode zur Klasse RoomClass; die Input- und Outputfunktionen sind gelb; die Berechnungen orange.

Es ist erkennbar, dass dieses Objekt auf dem **Systemobjekt** und dem **Hausobjekt** aufbaut, da hier die abstrakten Funktionen der Superobjekte durch die implementierten Funktionen konkretisiert wurden. Hier lassen sich die in der Tabelle festgelegten Funktionen wiederfinden.

Bei der Beschreibung der weiteren Hausobjekte **Wand** und **Heizgerät** wird der konkrete Quelltext in Java nicht mehr angegeben. Der gesamte Quelltext der SiMo-Umgebung ist auf der beiliegenden CD archiviert.

Das Haus-Objekt Wand

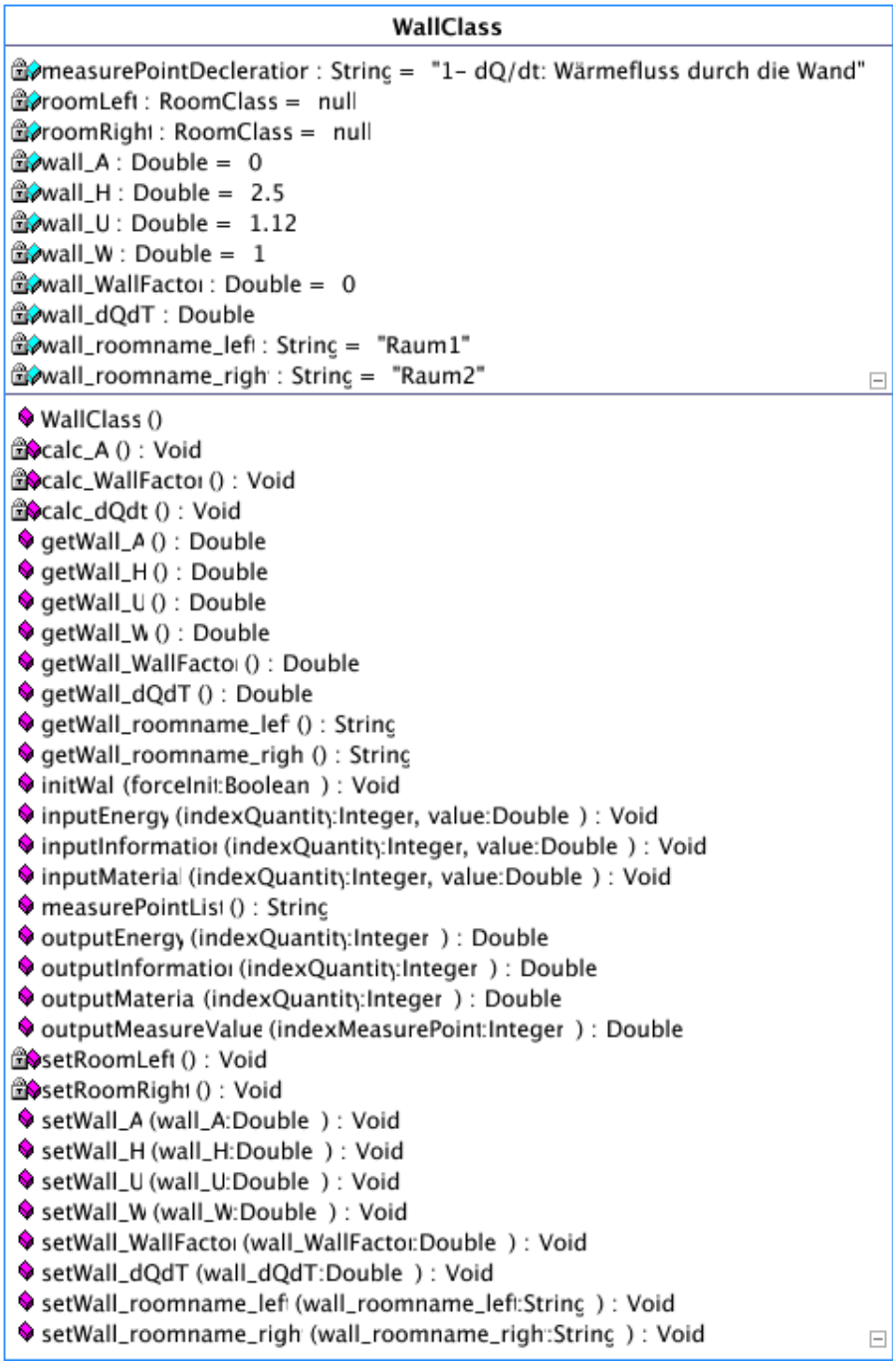


Abb. 22 UML-Darstellung der Wand-Klasse WallClass

Ein Vergleich dieser konkreten Klasse **WallClass** mit der in Abb. 9 zeigt, dass diese wieder größtenteils übernommen werden konnte.

Jetzt die Bindung der Wand an die beiden anliegenden Räume sichtbar. Die **WallClass** weist die zwei Eigenschaften **roomLeft** und **roomRight** auf. Dies sind die Verknüpfungen zu den beiden Raumobjekten, mittels derer die das Wandobjekt über die **outputInformation()**-Methoden der Raumobjekte die Temperatur der Räume abfragen kann.

Die **WallClass** besitzt wie die **RoomClass** eine Initialisierungs-Methode, um die Eigenschaften, die über die anderen Eigenschaften errechnet werden können, durch die Berechnungen zu initialisieren.

Die Berechnungen der Eigenschaften **Fläche A** und **WallFactor** und des **Wärmedurchgangs** werden über die **calc**_-Funktionen realisiert. Der Wandfaktor ist das Produkt aus U-Wert und Wandfläche. Er wird hier aus zwei Gründen gebildet. Erstens soll er in der Simulation mit angezeigt werden, um den User zu zeigen, dass der Wärmefluss durch die Wand hauptsächlich durch diesen Faktor bestimmt wird und zweitens zur Übersichtlichkeit des Quelltextes.

Die **inputEnergy()**–Methode bestimmt mit Hilfe der Raumtemperaturen (Methode **outputInformation()** von den Raumobjekten liefert die Werte) den Wärmefluss durch die Wand und setzt die Eigenschaft **Wärmefluss**.

Erst die **outputEnergy()**–Methode greift auf die verknüpften Raumobjekte zu und verändert über die **inputEnergy()**-Methoden der Raumobjekte die jeweiligen Wärmeinhalte der Räume.

Die weiteren Input- und Output-Methoden sind nicht belegt, da die Wand als energieumsetzendes System betrachtet wird.

Zu erwähnen sind noch die zwei Methoden **setRoomLeft()** und **setRoomRight()**. Sie setzen die Zeiger auf die Raumobjekte, und zwar mit Hilfe der die Raumnamen beinhaltenden Strings, die von den GUIs an die Objekte gesendet werden.

Der Quelltext für die Klasse **WallClass** kann auf der beiliegenden CD eingesehen werden.

7.3.3 Das Haus-Objekt Heizgerät

Das Heizgerät ist ein komplexeres Objekt, da in ihm vier verschiedene Heizgerätetypen realisiert worden sind. An diesem Objekt kann studiert werden, wie leicht sich die Komplexität von Objekten ohne Verletzung der allgemein gültigen Hüllen der Objekte steigern lässt. Abb. 23 zeigt die UML-Darstellung der programmierten Klasse **RadiatorClass**.

*Abb. 23 UML-Darstellung der Klasse **Radiator***

Werden die beiden UML-Darstellungen erneut miteinander verglichen, so lassen sich auch hier ein paar kleine Unterschiede feststellen. In der vorherigen Darstellung des Heizgerätes wurden für die vier verschiedenen Modi vier UML-Diagramme verwendet. Jetzt werden diese vier Modi in einer Klasse realisiert. Dies ist möglich, da es bei unserem Modell des Hauses nur auf die Inputs und Outputs der Systeme ankommt. Wie sich das Objekt im Inneren verhält, ist für die Interaktion der Objekte (Subsysteme) irrelevant.

Da alle vier Modi mit einer Klasse realisiert werden, müssen natürlich alle Eigenschaften der vier Heizgerätetypen von Anfang an bereitstehen. Ob sie verwendet werden, hängt dann von der Wahl des jeweiligen Gerätetyps ab. Ebenso verhält es sich mit den Methoden.

Die Entscheidung, welcher Heizgerätetyp verwendet werden soll, wird mit dem Setzen des Heizgerätemodus durch die Funktion **setRadiatorModus()** getroffen. Dies geschieht durch den User über das GUI, das Zugriff auf diese Methode hat.

Die einzige Methode, die Auswirkungen auf andere Haus-System-Objekte hat, ist die Methode **outputEnergy()**. Hinter dieser Methode steckt eine Methodenstruktur, die die unterschiedlichen Modi verwaltet. In Abb. 24 ist diese Struktur dargestellt.

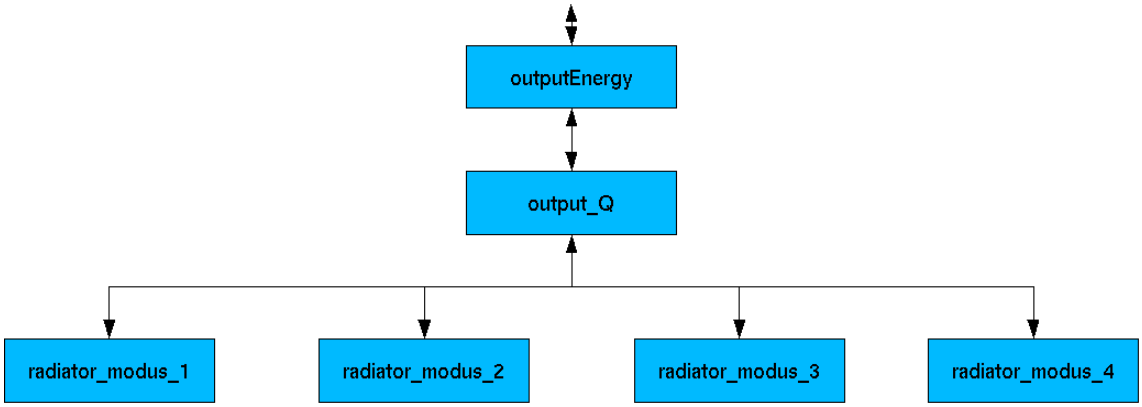


Abb. 24 Struktur der Methode outputEnergy()

Es wird ersichtlich, dass die Methode **outputEnergy()** ihren Outputwert von der Methode **output_Q()** anfordert. Die Methode **output_Q()** wiederum fordert ihren Wert, je nach gewähltem Heizgerätemodus, von den Methoden **radiator_modus_1()** bis **radiator_modus_4()** an. In den Methoden **radiator_modus_1()** bis **radiator_modus_4()** sind die Verhaltensweisen der Heizgeräte hinterlegt. Die Verhaltensweisen der Heizkörper wurden in Kapitel 7.2.3 beschrieben.
Der Quelltext befindet sich auf der CD in der Klasse **RadiatorClass** unter den Methoden **radiator_modus_1** bis **radiator_modus_4**.
Die restlichen Input- und Output-Methoden sind nicht belegt.

7.3.4 Die Modellierungs-Umgebung

Nachdem das System **Haus** vollständig in Java abgebildet wurde, wird jetzt die Modellierungs-Umgebung entwickelt, die dem User erlaubt, aus den Objektdefinitionen (Systemdefinitionen) ein eigenes Modell des Systems **Haus** mit den vordefinierten Haus-Objekten (Haus-Subsystemen) zu modellieren. Die Aufgabe der Modellierungsumgebung ist es, Räume, Wände und Heizgeräte zu erzeugen und zu konfigurieren. Dazu werden die schon erwähnten GUIs (General User Interfaces) benötigt. Sie sind im Prinzip nichts anderes als Möglichkeiten Eingaben in das Programm zu machen. Diese Eingaben können je nach Anspruch und Ziel des Programms mehr oder weniger komfortabel ausfallen. Da die SiMo-Umgebung die Erlangung technischer Handlungskompetenz fördern soll, fällt hier die Komfortabilität dementsprechend gering aus. Die Eingaben sollen überlegt geschehen und Konsequenzen haben, wenn sie falsch oder unüberlegt sind. In diesem Fall bedeutet es, dass das Programm falsche Werte liefert oder abstürzt.
Da die GUIs keine Hilfen zur Eingabe liefern, und die Eingaben nicht auf Plausibilität und Richtigkeit geprüft werden, verhält sich das Programm wie oben beschrieben.
Die GUIs zu den Haus-Objekten werden in den folgenden Objekten kurz vorgestellt. Die notwendigen Eingaben können dem User's-Manual zum Programm **HausSim** entnommen werden.

Die Raum-GUI

Das GUI zum RaumObjekt wird in Abb. 25 dargestellt.

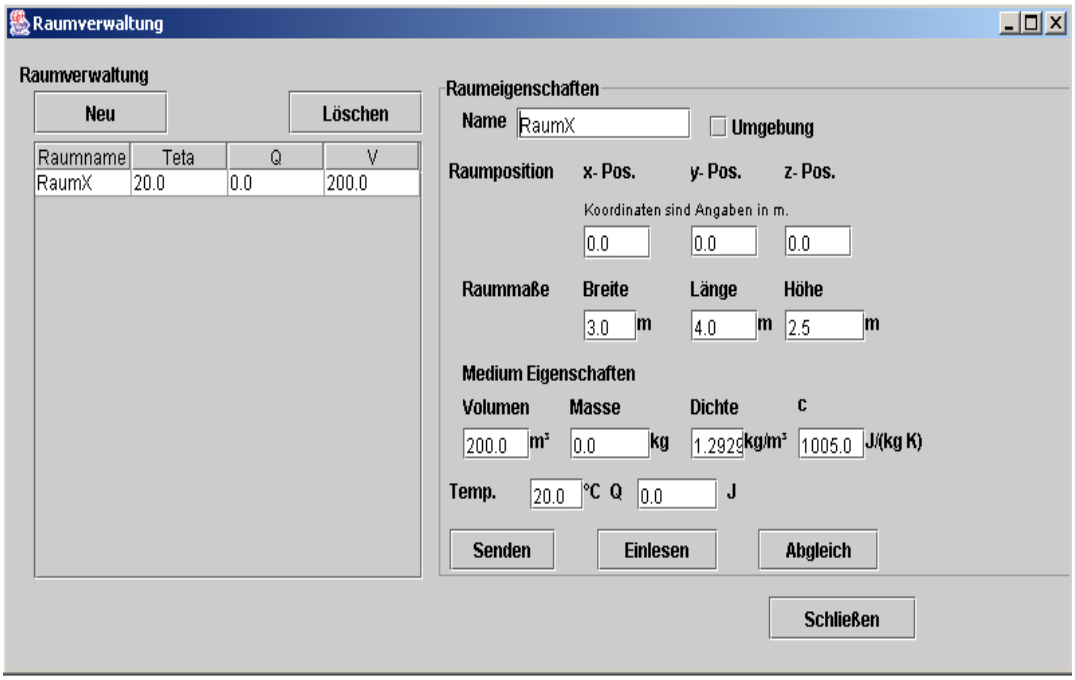


Abb. 25 GUI für die Raumobjekte

Hier können Räume generiert, konfiguriert und gelöscht werden. Die linke Seite der Eingabe dient zur Übersicht über die bestehenden Raum-Objekte. Die rechte Seite erlaubt die Manipulation der über die linke Tabelle ausgewählten Raum-Objekte. Die wichtigsten Methoden dieser und der weiteren GUI-Klassen sind die, welche die Eingaben an die zu konfigurierenden Objekte senden und die, welche die Werte der Eigenschaften der Methoden auslesen und in der GUI darstellen. Diese Methoden werden in den GUIs einheitlich **datenSenden** und **datenLesen** genannt. Sie transformieren die Eingabewerte in den entsprechenden Datentyp der Eigenschaft und schreiben ihn über die Setter-Methoden in die Eigenschaft der jeweiligen Objekte ein. Umgekehrt werden die Werte der Eigenschaften der Objekte über die Getter-Methoden eingelesen und vom Original-Datentyp in den String-Typ umgewandelt.

Wand-GUI

Die Wand-GUI ist in Abb. 26 dargestellt.

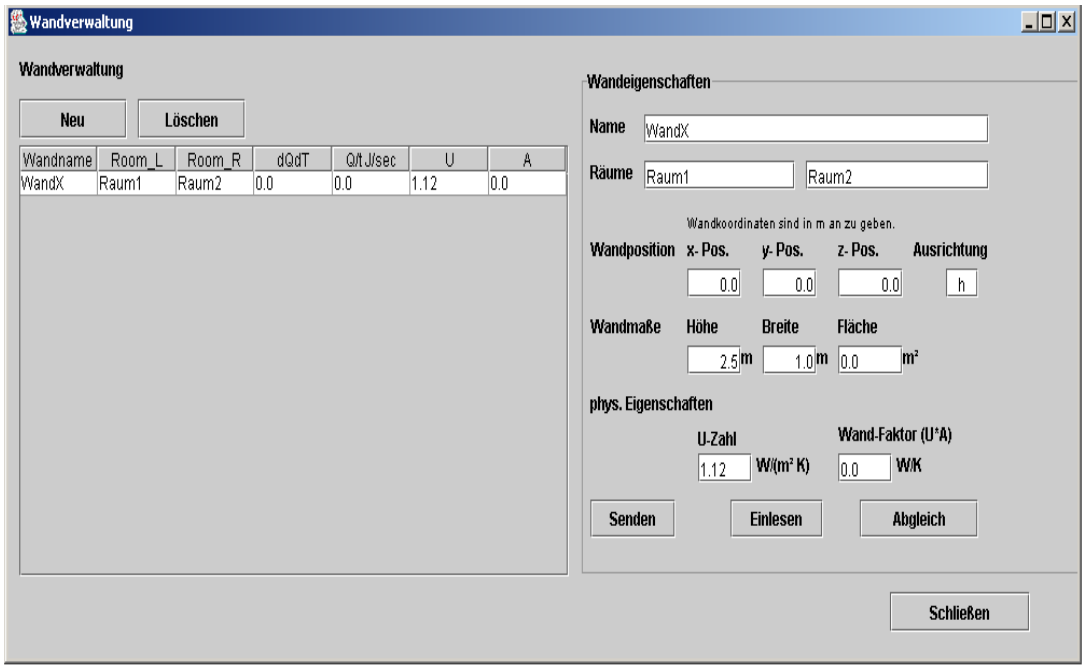


Abb. 26 GUI für die Wandobjekte

Hier gelten die gleichen Erklärungen wie für das Raumobjekt.

Heizgeräte-GUI

Die Heizgeräte-GUI wird in Abb. 27 und Abb. 28 dargestellt. Bei den Heizgeräten gibt es eine Staffelung der Eingabe. Es existiert ein Fenster, in dem die Heizkörper verwaltet werden. Über den Button **Heizkörper konfigurieren** wird ein weiteres Fenster geöffnet, in dem der ausgewählte Heizkörper konfiguriert werden kann (s. Abb. 27).

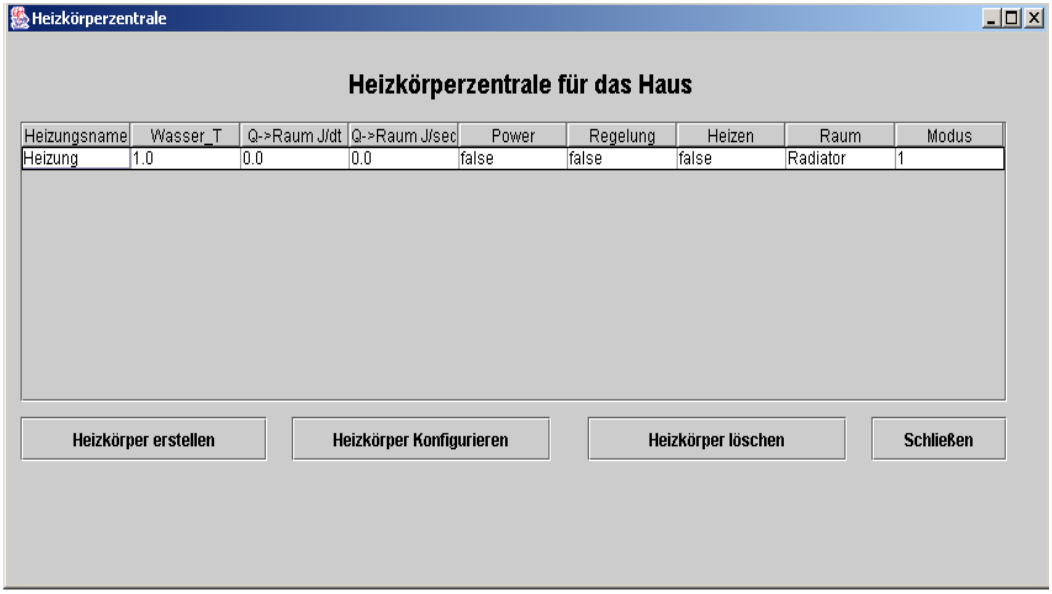


Abb. 27 GUI für die Erstellung, Konfiguration und Löschung von Heizgeräteobjekten

Im GUI zur Konfiguration der Heizgeräte Abb. 27 wird ersichtlich, wo der User die Möglichkeiten zur Bestimmung des Heizgerätemodus hat. Die entsprechenden Eigenschaften für das Heizgerät werden erst nach der Wahl des Modus für das Heizgerät relevant. Die Zuordnung der Eigenschaften zu den verschiedenen Modi ist aus der Beschreibung des Heizgerätesystems in Kapitel 7.2.3 ersichtlich.

Abb. 28 GUI Konfiguration des ausgewählten Heizgerätes

7.3.5 Die Simulationsumgebung

In der Simulationsumgebung sind die Objekte **Rechenmodell**, **Timer Objekt-container** und die **Messgeräte** zusammengefasst. Von hier aus wird das Haussystem gesteuert und überwacht. Die einzelnen Objekte werden zur Interaktion angeregt und kontrolliert. Dazu sind die in den nächsten Kapiteln beschriebenen Objekte notwendig.

Das Rechenmodell

Das Rechenmodell ist die Steuerung der Simulation. Es gibt die Reihenfolge der Berechnungen der Objekte vor, stellt die zu aktivierenden Objekte bereit, sorgt für die Datenausgabe der Hausobjekte an die GUIs und für die Darstellung der Messpunkte der Hausobjekte in den Messgeräten. Abb. 29 zeigt den chronologischen Ablauf der Simulationsschritte.

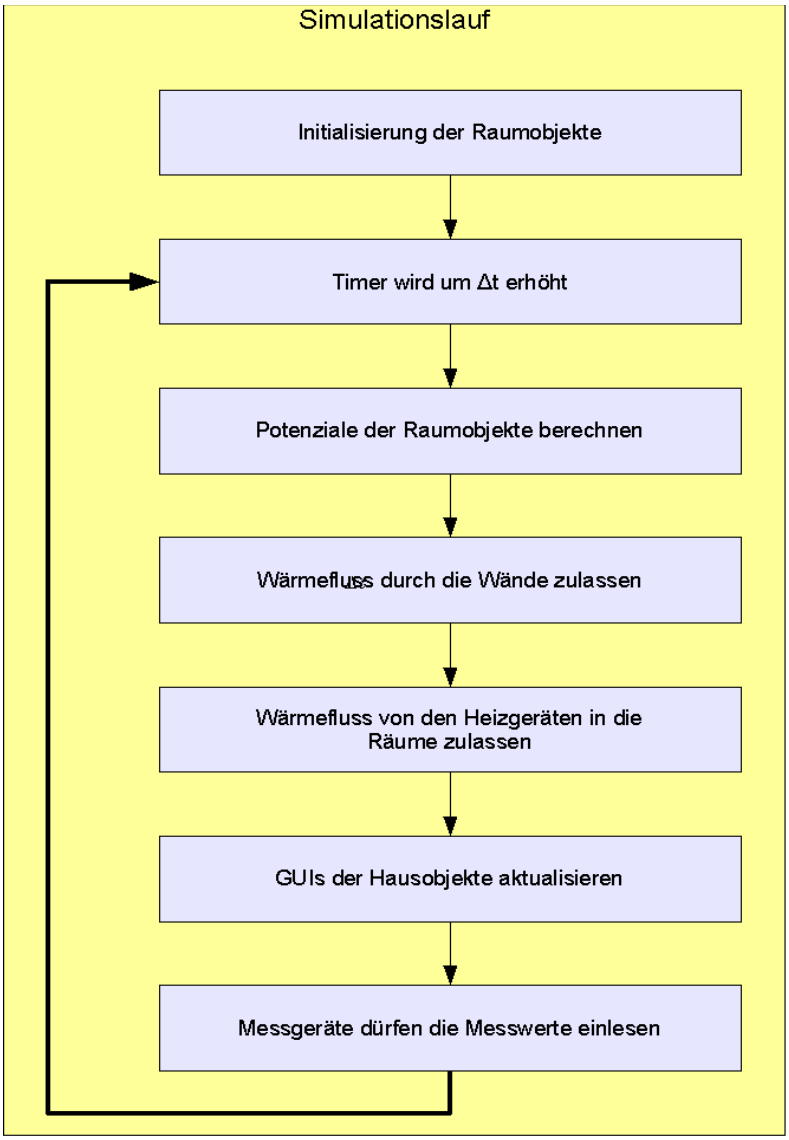


Abb. 29 Ablaufdiagramm der Simulation

Aus dem Ablaufdiagramm (Abb. 29) wird ersichtlich, dass es sich bei dem Simulationsablauf um eine Schleife handelt. Aufgrund der physikalischen Gegebenheit, dass ein so komplexes System wie das Haus zu jedem Zeitpunkt nur als Ganzes betrachtet werden kann, sind keine eigenständigen Aktionen der Hausobjekte erlaubt. Sie dürfen erst dann in Aktion treten, wenn dies durch das Rechenmodell induziert wird. In Abb. 29 ist außerdem ersichtlich, dass auch der Timer durch das Rechenmodell gesteuert wird.

Der Timer

Der Timer ist notwendig, damit der Zeitpunkt der einzelnen Zustände der Hausobjekte eindeutig ist. Der Timer ist ein Zähler, der die Zeit in den vorgegebenen Zeitabständen **Δt** weiterzählt. In Abb. 30 ist das GUI des Timers zu sehen.

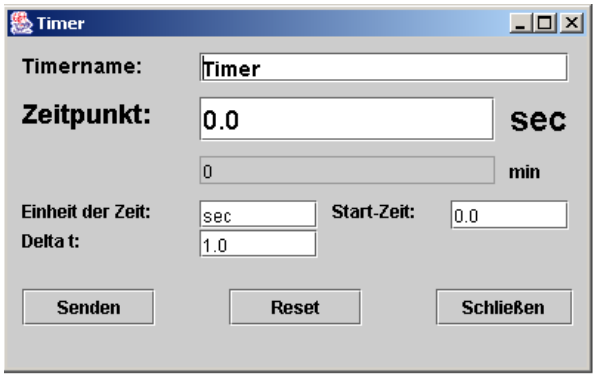


Abb. 30 GUI des Timer-Objektes

Die Objektcontainer

Die Objektcontainer sind ein wesentlicher Bestandteil der Simulationsumgebung; ohne sie ließen sich keine Zugriffe auf die Objekte der Simulation realisieren. In den Objektcontainern sind alle Objekte in Form von Tabellen hinterlegt, damit sie über ihre Objektnamen angesprochen werden können. Aus diesem Grund muss der vergebene Name der Objekte eindeutig sein, da die Container sonst keine Möglichkeit haben, das gewünschte Objekt auch zur Verfügung zu stellen. Es gibt in der Simulationsumgebung mehrere Objektcontainer, die jeweils einen Objekttyp verwalten. Abb. 31 zeigt die Hierarchie der Objektcontainer.

In Java ließen sich zwar alle Objekte in einem Container verwalten, dies hätte jedoch während der Simulation zu große Nachteile. Die Simulation greift direkt auf die Container der benötigten Objekte zu. Da die Berechnungen für die ein-

zelnen Objekte eines Systemtyps keiner bestimmten Reihenfolge unterliegen müssen, beginnt die Simulation die Berechnung beim ersten Objekt und hört beim letzten auf. Dies verkürzt die Berechnungszeit der Objekte, da nicht erst die Objekte eines Typs gesucht werden müssen. Die Simulation ruft also die Container der Reihe nach auf und berechnet strikt die Objekte in ihrer abgelegten Reihenfolge.

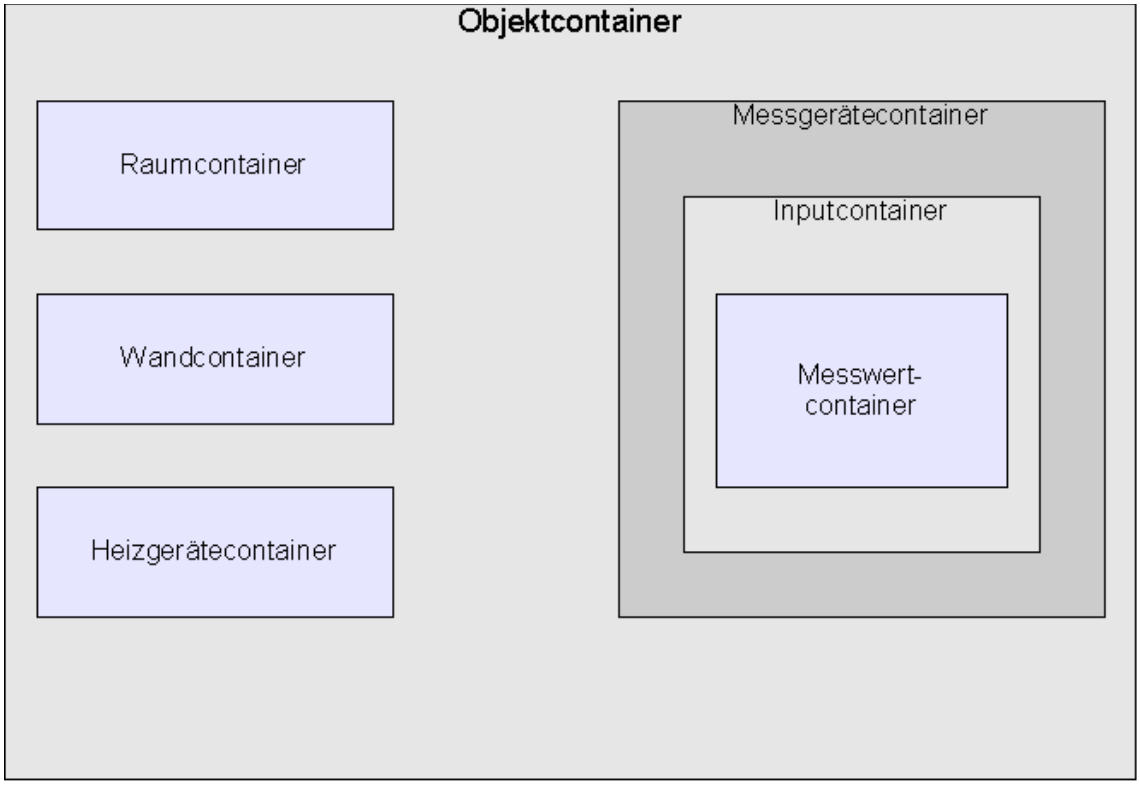


Abb. 31 Hierarchie der Objektontainer

Die Messgeräte

Die Messgeräte ermöglichen die grafische Darstellung der berechneten Werte. Ihre Konzipierung war sehr aufwändig, da sie sehr hohen Ansprüchen genügen, die sich sowohl auf den Einsatz in der Simulation als auch auf die Einbindung in die Programmierung beziehen.

Anforderungen an die Messgeräte im Simulationseinsatz.

1. Der User kann die Anzahl der Messgeräte, die sein System überwachen sollen, selbst bestimmen.
2. Der User hat die umfassende Möglichkeit, die für ihn wichtigen Werte des Hauses zu erfassen.
3. Der User kann den Darstellungsbereich der Messwerte selbst definieren.
4. Der User hat die Möglichkeit, mehrere Input-Größen zu einem X-Wert in einem Graphen darzustellen.
5. Der User kann die Größe für den X-Input selbst festlegen.
6. Die Bedienung orientiert sich an der eines modernen Speicher-Oszilloscops.
7. Der User kann selbst entscheiden, in welchem **Δx** er die Messwerte erfassen möchte. Die Genauigkeit der Berechnungen lässt sich über die Wahl von kleinen Zeitabschnitten (**Δt)** wesentlich steigern. Aufgrund des begrenzten Messwertspeichers ist es erforderlich, den Simulationstimer von der Messwerterfassung zu entkoppeln, um dennoch eine aussagekräftige Darstellung zu gewährleisten.

Auf der Basis dieser Anforderungen wurde ein Messgerät mit folgenden Features entwickelt.

- frei wählbarer Darstellungsbereich der X- und Y-Achse
- frei wählbares Datenerfassungsintervall für das gesamte Messgerät
- acht frei konfigurierbare Eingänge (1x X-Input und 7x Y-Inputs)
    - der X-Input-Kanal und
    - die sieben Y-Input-Kanäle können mit jeder beliebigen Größe belegt werden
    - Jeder Y-Input kann mit einer eigenen Farbe, Verstärkungsfaktor (dient zum Ausgleich, wenn zwei Größen mit sehr unterschiedlichen Größenordnungen dargestellt werden sollen), Speicherintervall der Messwerte, Darstellungsintervall und der Speichergröße konfiguriert werden.

Abb. 32 zeigt das GUI zur Input-Konfiguration.

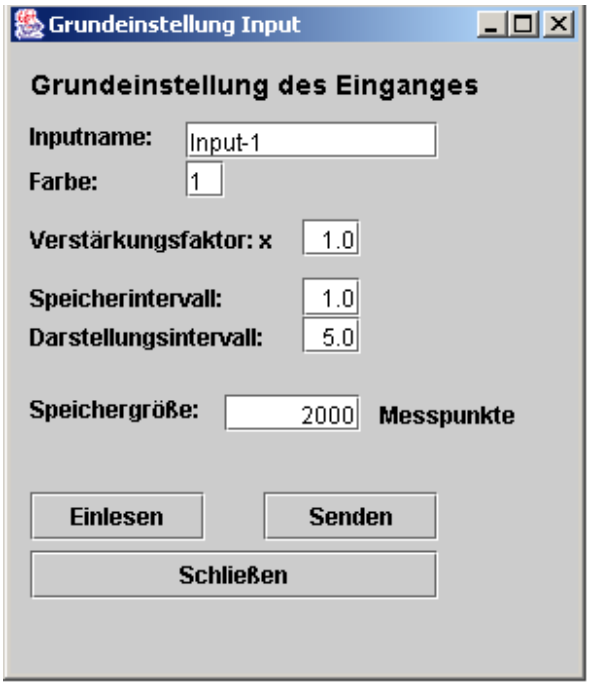


Abb. 32 GUI der Konfiguration eines Y-Inputs

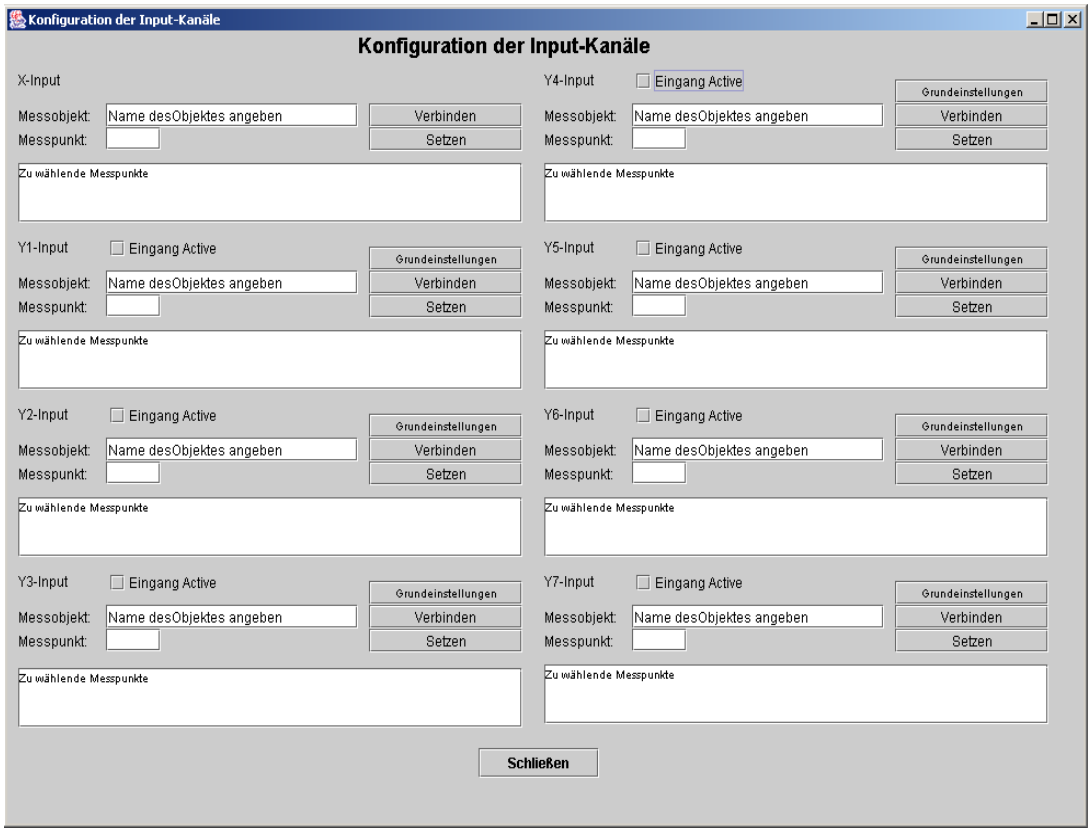


Abb. 33 GUI zur Konfiguration der einzelnen Input-Kanäle

In Abb. 33 wird das GUI zur Festlegung der einzelnen Input-Kanäle dargestellt.

Mit diesen Eigenschaften sind die Messgeräte in der SiMo-Umgebung universell verwendbar. Details zur Bedienung können dem User's Manual zur SiMo-Umgebung entnommen werden.

Das Messgerät zur Verwendung in der Programmierung

Um dem User während der Programmlaufzeit ein bedienerfreundliches und dennoch universelles Messgerät für seine Systemanalysen bereitzustellen, wird ein spezielles Messkonzept benötigt. Das hier neu entwickelte Messkonzept entspricht den oben genannten Anforderungen und ermöglicht dem Programmierer außerdem eine unkomplizierte Implementierung des Messgeräteobjektes in eine beliebige Systemumgebung. Das Messkonzept basiert auf wenigen Vereinbarungen, die während der Programmierung eingehalten werden müssen.

Das Messkonzept ist in Abb. 34 dargestellt. Es wird ersichtlich, dass die Systemobjekte über ein Interface mit den Input-Kanälen des Messgerätes verknüpft sind. Während der Programmierphase wird dieses Interface lediglich an alle zu messenden Systemobjekte angehängt. Diese können dann von den Input-Kanälen des Messgerätes erkannt werden und während der Laufzeit der Simulation zur Auswahl gestellt werden. Die Realisierung wird im folgenden Kapitel erläutert.

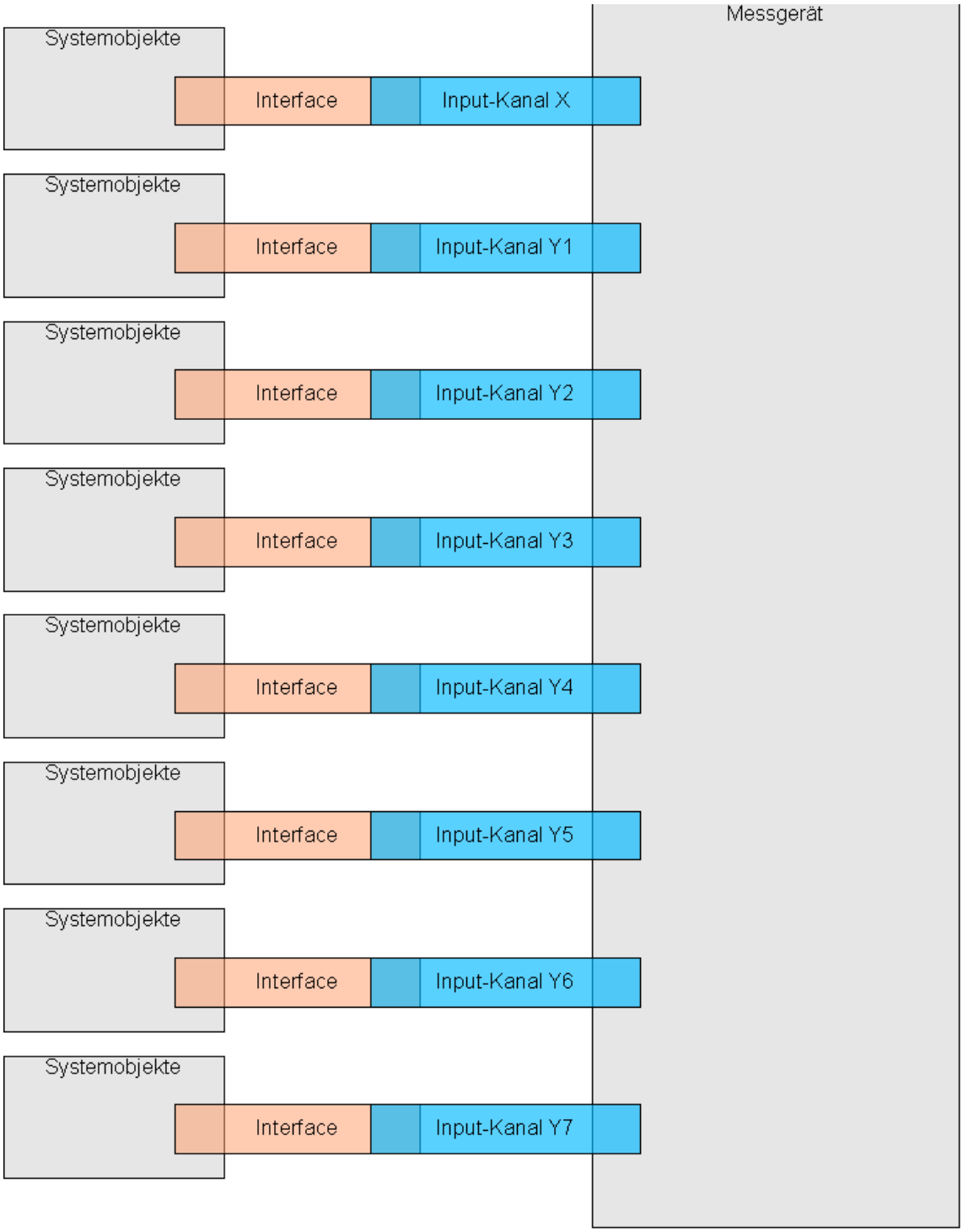


Abb. 34 Systemschaltbild des Messkonzeptes

7.3.5.1 Das Messkonzept der Simulationsumgebung

Das Messkonzept ist in Abb. 34 dargelegt. Daraus wird ersichtlich, dass die zu messenden Systemobjekte über ein Interface verfügen, das Daten an die Input-Kanäle des Messgerätes weiterleitet. Die einzelnen Input-Kanäle des Messgerätes übernehmen dann die weitere Verarbeitung der Messdaten.
Im Folgenden wird der Aufbau und die Funktion des Interfaces beschrieben.
Das Interface stellt eine Schnittstelle bereit, über die der Input-Kanal des Messgerätes die nötigen Messinformationen des Systemobjektes abfragen kann. Das Interface stellt eine Art abstrakte Klasse dar. Sie kann vom Input-Kanal des Messgerätes angesprochen werden. Die Informationen werden anschließend vom Systemobjekt via Interface abgefragt. Das Interface muss nicht den Objekttyp des zu messenden Systems kennen, sondern lediglich die bereitgestellten Daten. Das Interface ist in Abb. 35 als UML-Darstellung zu sehen.

Die Methode **measuredPointList()** liefert eine Liste aller Messpunkte des zugehörigen Systems. Diese ist notwendig, damit dem User während der Laufzeit der Simulation eine Auswahl der von den Systemobjekten angebotenen Messpunkte bereitgestellt werden kann.
Der Input-Kanal des Messgerätes holt sich die zu messenden Daten über die Methode **outputMeasureValue(indexMeasurePoint)**. Dabei muss der Parameter **indexMeasurePoint** aus der Messpunkt-Liste übergeben werden.

Die oben beschriebenen Funktionalitäten müssen in dem zu messenden Systemobjekt implementiert sein. Die UML-Darstellungen in den Abbildungen der Objekte, des Interfaces **MeasuredObjekt** (Abb. 36) und des Raumobjektes **RoomClass** (s. Abb. 35) - stellvertretend für die anderen Haussystemobjekte - verdeutlichen den Zusammenhang. In beiden Klassen finden sich die oben genannten Methodennamen wieder.

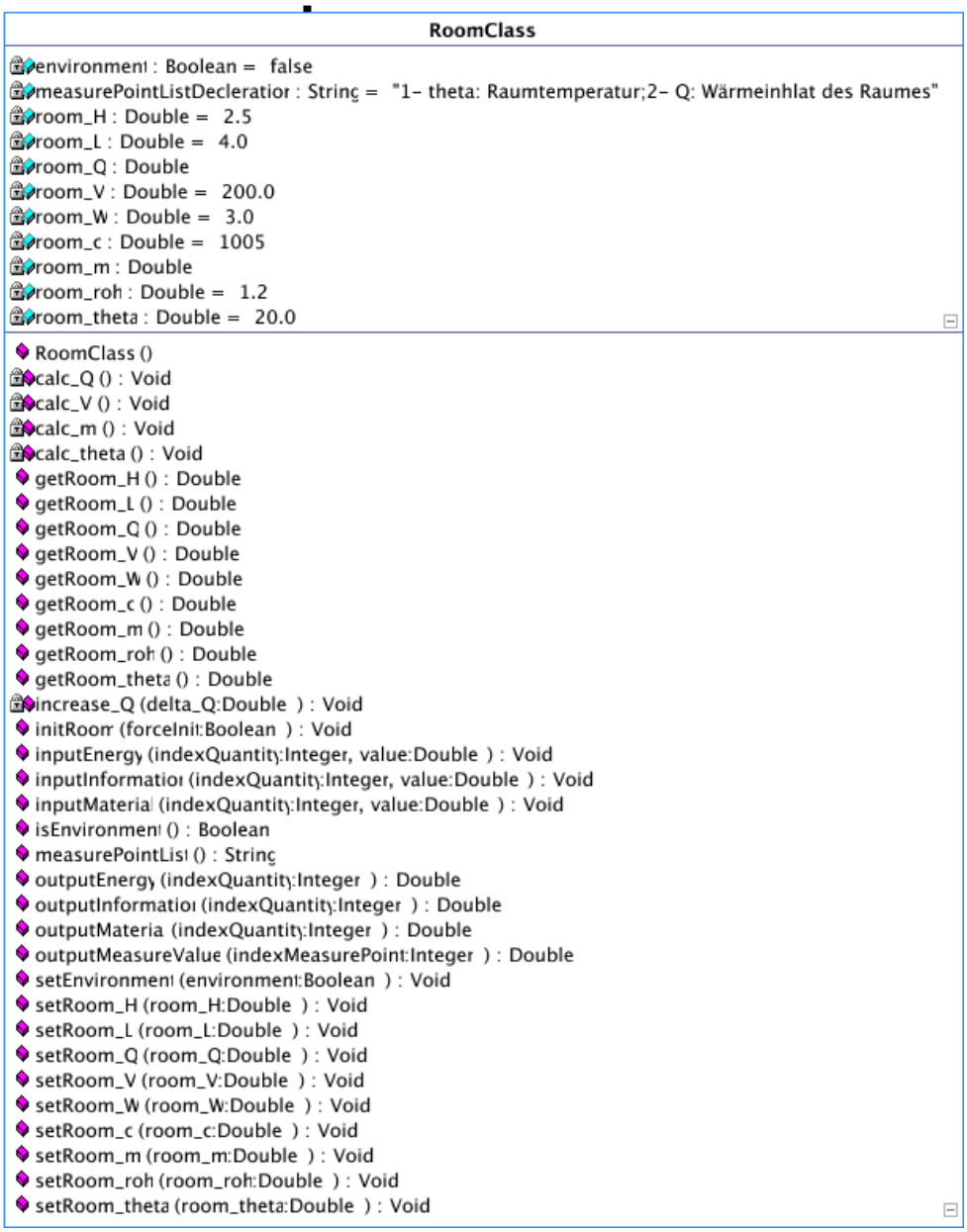


Abb. 35 UML-Darstellung des Objektes RoomClass

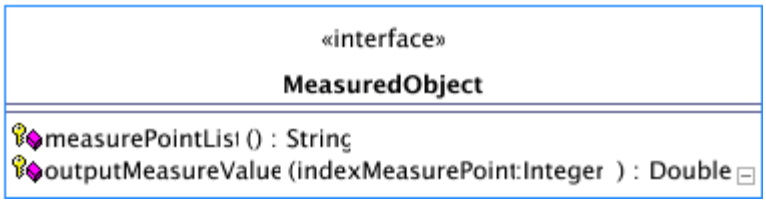


Abb. 36 UML- Darstellung des Interfaces MeasuredObject

Die Umsetzung der beiden Methoden zur Messwerterfassung in der Klasse **RoomClass** wird in den Abb. 37 und Abb. 38 als Aktivitätsdiagramm in UML dargestellt. Es handelt sich dabei lediglich um eine Stringausgabe des Strings, der die Messpunkte aufführt (String: 1- Raumtemperatur; 2- Wärmeinhalt in dem Raum) und um eine Selektionsschleife, die bestimmt, welcher Wert des Objektes ausgegeben werden soll. Dabei ist der zu übergebende Parameter die Ziffer vor der Messgröße aus der String-Liste.

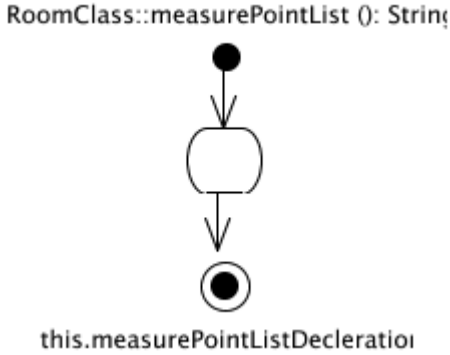


Abb. 37 Methode messliste()

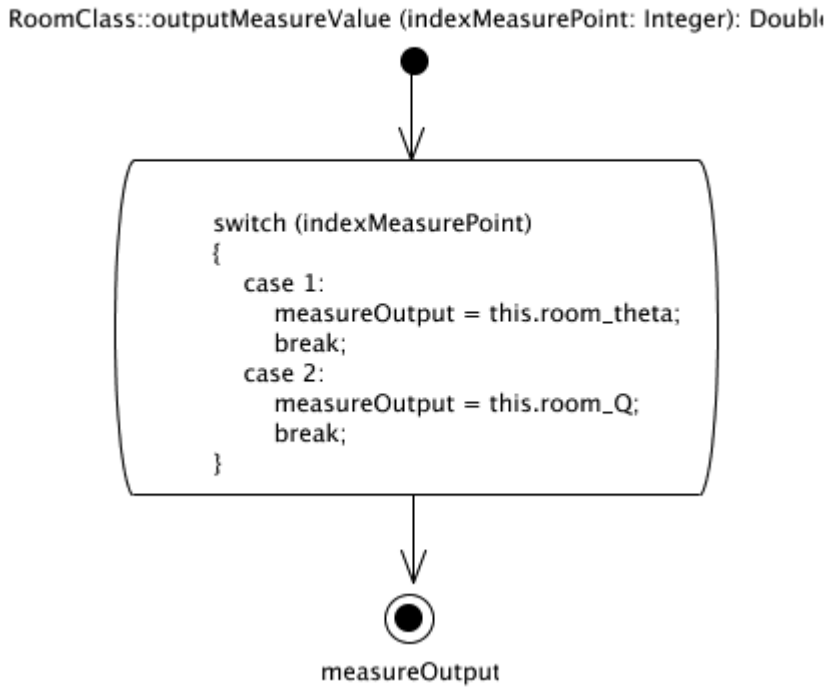


Abb. 38 Methode outputMeasureValue(...)

Der Aufbau des Messgerätes

Der prinzipielle Aufbau des Messgerätes wird nun beschrieben. Wie aus den vorherigen Darstellungen zur Programmierung von Java erkennbar geworden ist, kann man nun aus dem Systemschaltbild des Messgerätes in Abb. 39 und der dazugehörigen Beschreibung eine Vorstellung bekommen, wie die Objekte mit ihren Eigenschaften und Funktionen aussehen
(s. CD). Das Messgerät besteht aus mehreren Subsystemen.

Das GUI des Messgerätes

Das GUI dient zur Konfiguration des Messgerätes, der Monitordarstellung (Koordinatensystemeinstellungen), der Belegung der Input-Kanäle und zur Konfiguration der einzelnen Input-Kanäle. Die Bedienung des GUI kann im User's Manual nachgelesen werden.

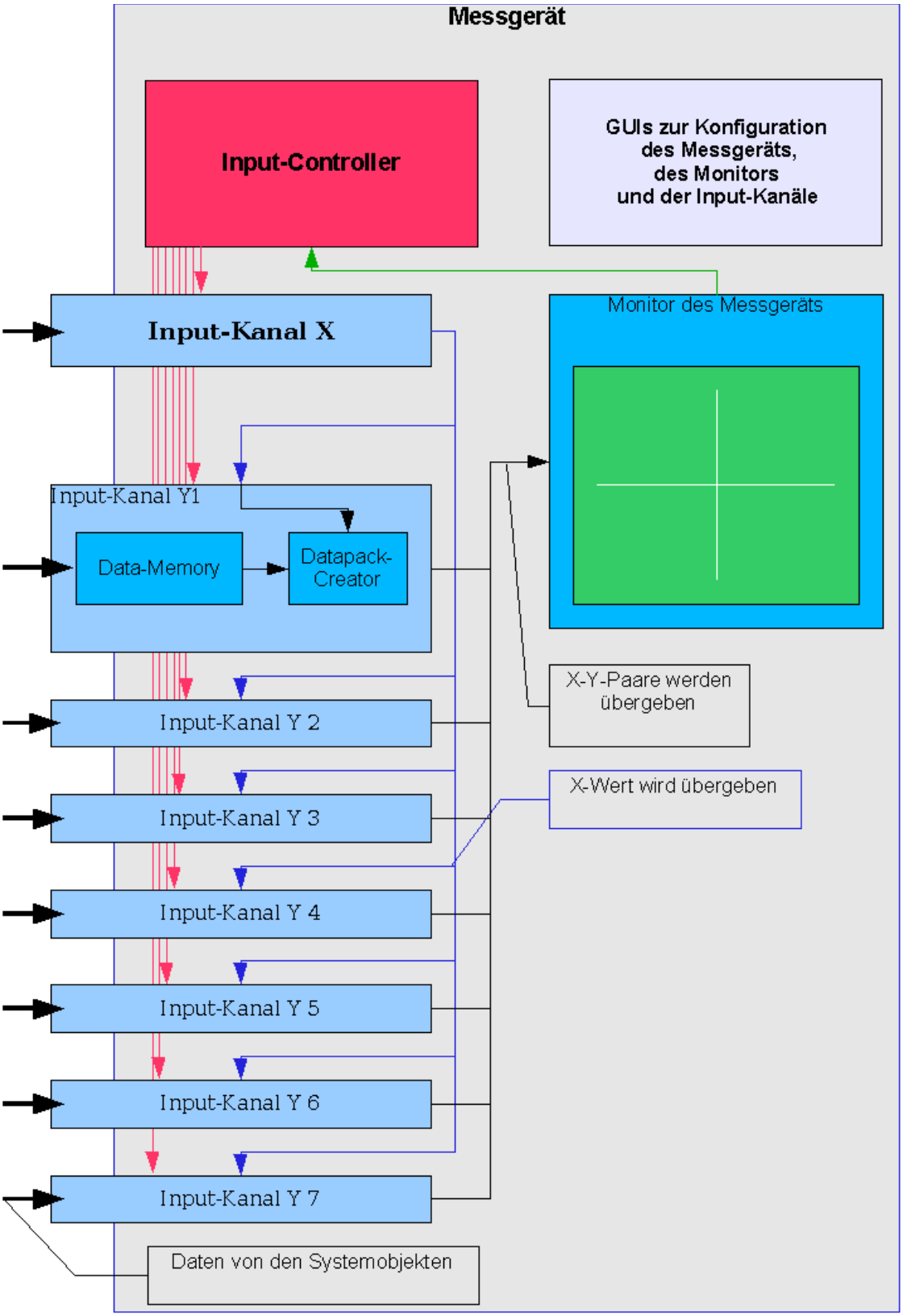


Abb. 39 Systemschaltbild des Messgerätes

Der Monitor des Messgerätes

Der Monitor des Messgerätes dient zur Darstellung der Messdaten aus den sieben Y-Input-Kanälen. Diese werden als Datenpunkte übermittelt. Während der Laufzeit der Simulation kann der User den Monitor über das GUI konfigurieren.

Der Input-Controller

Der Input-Controller hat die Aufgabe, die einzelnen Input-Kanäle zu steuern und anzusprechen. Seine Aktivierung erfolgt über das Messgerät. Er gibt den Inputs Signale, damit sie zum richtigen Zeitpunkt Daten anfordern.

Zur Datenausgabe auf dem Monitor fordert dieser über den Controller die gespeicherten Daten der einzelnen Y-Input-Kanäle an, welche diese direkt an den Monitor senden.

Die Input-Kanäle des Messgerätes

Die Input-Kanäle sind die direkte Verbindung zu den Interfaces der zu messenden Systemobjekte. Sie sind in der Lage, die Messpunktlisten anzufordern und die Daten von den ausgewählten Messpunkten abzufragen. Die Messpunktlisten werden dem GUI zur Verfügung gestellt, damit dieses dem User die Messpunkte zur Auswahl anbieten kann.

Es gibt zwei Arten von Input-Kanälen. Den X-Input und die sieben Y-Inputs.

Der **X-Input** dient nur zur Anforderung des X-Wertes. Dieser Wert wird zur Weiterverarbeitung an die sieben Y-Inputs gesendet.

Die **Y-Inputs** selbst sind wieder komplexe Subsysteme. Sie fragen die Messwerte von den Systemobjekten ab. Aus dem eigenen Messwert und dem Messwert aus dem X-Input wird ein Messpunktobjekt generiert. Dieses Messpunktobjekt wird im Data-Memory abgespeichert. Die Größe des Datenspeichers wird durch die Vorgabe des Users beschränkt. Es handelt sich dabei um ein FIFO-Register (First In First Out) mit Speicherfunktion. Bei der Speicher-

funktion handelt es sich auch um einen FIFO-Typen, d. h. bei vollem Speicher wird der erste Wert aus dem Speicher geschoben und der neue Wert hinten angehängt.

Diese Messpunkte werden dann durch den Datapack-Creator manipuliert, so dass sie auf dem Monitor darstellbar sind. Die Manipulation wird durch den User über das GUI bestimmt (Farbe der Messpunkte, Verstärkung). Aus dem manipulierten Messwertpunkt und der Zusatzinformation der Darstellungsfarbe wird ein Datapack generiert, das an den Monitor gesendet wird. Dieser kann dieses Datapack ohne weitere Bearbeitung sofort auf dem Screen darstellen.

Anmerkung zum Messgerät

Die Darstellung des Aufbaus des Messgerätes verlässt die strikte Anwendung der in der Systemtheorie verwendeten allgemeinen Begriffe Input und Output für die Funktionalitäten. Hier werden die Methoden direkt nach ihrer Funktionalität benannt. Dies ist sinnvoll, damit der Programmierer die Funktionalität des Messgerätes besser nachvollziehen kann, da der funktionale Aufbau des virtuellen Messgerätes im Wesentlichen dem eines realen entspricht.

Diese weitere Beschreibung eines technischen Systems aus einer konkreteren Sicht auf ein System zeigt, wie flexibel und leistungsstark die objektorientierten Programmiersprachen sind, um reale Systeme nachzubilden. Dies ist ein weiterer Grund, im Zusammenhang mit technischer Bildung das Erlernen einer OOP zu fördern oder gar zu fordern.

8 Anwendungsbeispiele für die SiMo-Umgebung

8.1 Beispiel 1: Ein-Raum-Haus

In diesem Beispiel wird ein Ein-Raum-Haus simuliert.

Annahme: Das Haus besteht aus vier Wänden und einem Dach. Die Beschaffenheit der insgesamt fünf Wände (4 Wände + 1 Dach) ist gleich. Die Wände sind 3 m hoch. Es wird weiter angenommen, dass durch den Fußboden keine Wärme entweichen kann. Die weiteren Daten zum Haus können aus den Tabellen und Abb. 40 entnommen werden.

Fläche der Wände:	$A = 1 \cdot (10 \cdot 5\,m^2) + 2 \cdot (10 \cdot 3\,m^2) + 2 \cdot (5 \cdot 3\,m^2) = 140\,m^2$
Volumen des Raumes:	$V = 10 \cdot 5 \cdot 3 = 150\mathrm{m}^3$

Weitere Werte werden angenommen:

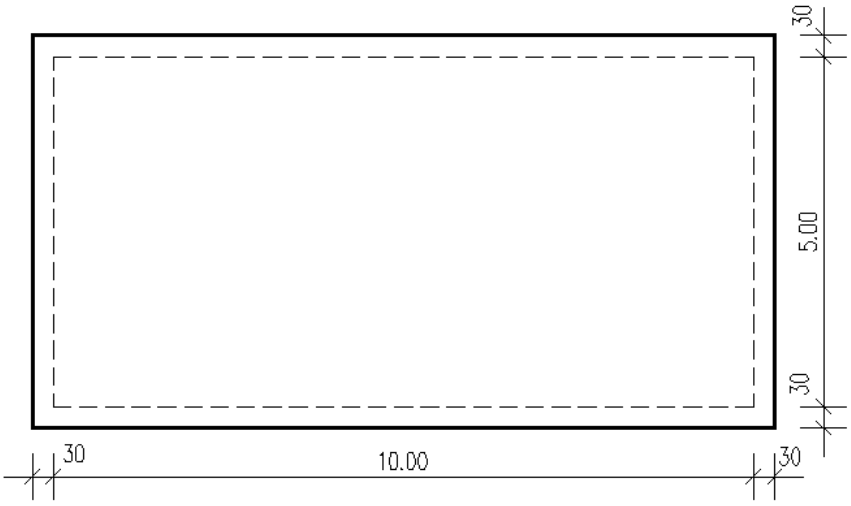
Normdichte der Luft bei 20°C u. 1013hPa:	$\varrho = 1{,}204 \frac{kg}{m^3}$
spezifische Wärmekapazität der Luft bei 20°C :	$c_p = 1{,}005 \frac{kJ}{kg \cdot K}$
Zimmertemperatur:	$\vartheta = 20\,°C$
Umgebungstemperatur	$\vartheta = 5\,°C$
Wärmedurchgangskoeffizient Gasbetonstein $1000 \frac{kg}{m^3}$ Dicke $30\,cm$ Außenwand	$U = 1{,}17 \frac{W}{m^2 \cdot K}$
Dichte Wasser (Heizkörper) bei 50°C und 1013,25 hPa	$\rho = 0{,}98804 \frac{g}{cm^3}$

Werte entnommen aus [Kuchling 1994].

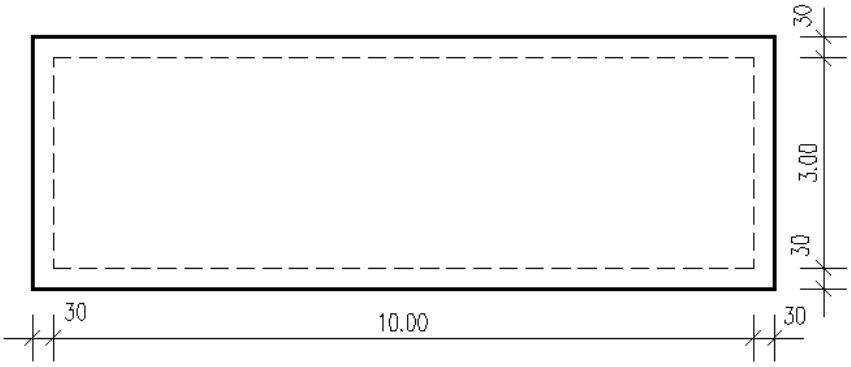
Bemerkung zur Tabelle:
Bei der Annahme dieser Werte werden Ungenauigkeiten toleriert. Da es sich bei dem Ein-Raum-Haus um ein dynamisches System handelt, müssten sich alle temperaturabhängigen Größen zu jedem Zeitpunkt mit der Temperatur ändern. Dieses wird aber aus didaktischen Gründen vernachlässigt. Bei den ausgewählten Werten handelt es sich um Beispielwerte, die aus den nicht fachspezifischen Standardformelsammlungen entnommen werden

Auf der nachfolgenden Seite zeigt Abb. 40 die Ansichten des Ein-Raum-Hauses.

Draufsicht



Vorderansicht



Seitenansicht

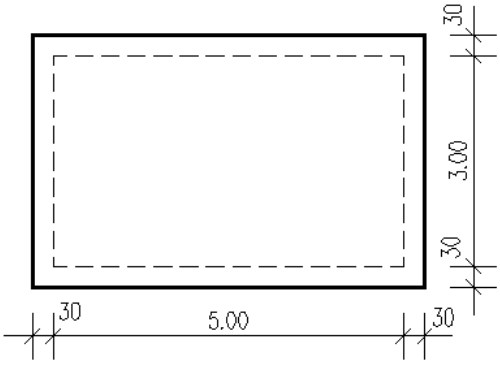


Abb. 40 Ansichten des Ein-Raum-Hauses

Mit den oben zusammengestellten Werte kann nun mit Hilfe der SiMo-Umgebung ein Ein-Raum-Haus modelliert werden.

Nach dem Starten der SiMO-Umgebung können die Subsysteme des Hauses definiert werden. Die folgenden Screenshots zeigen die GUIs zur Dateneingabe der Hausobjekte (Abb. 41, 42, 43).

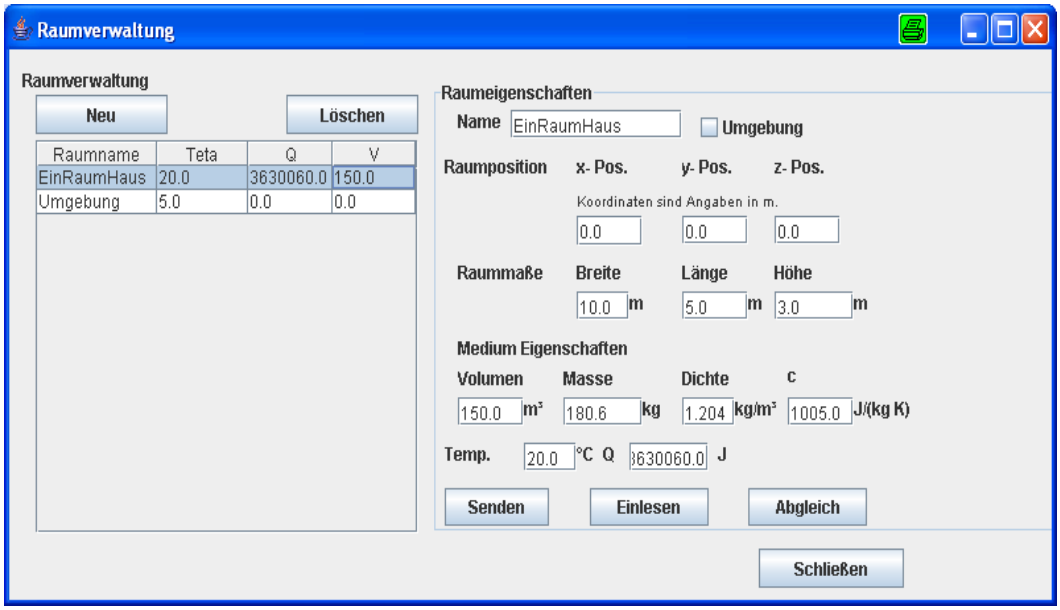


Abb. 41 Gui der Raumeingabe

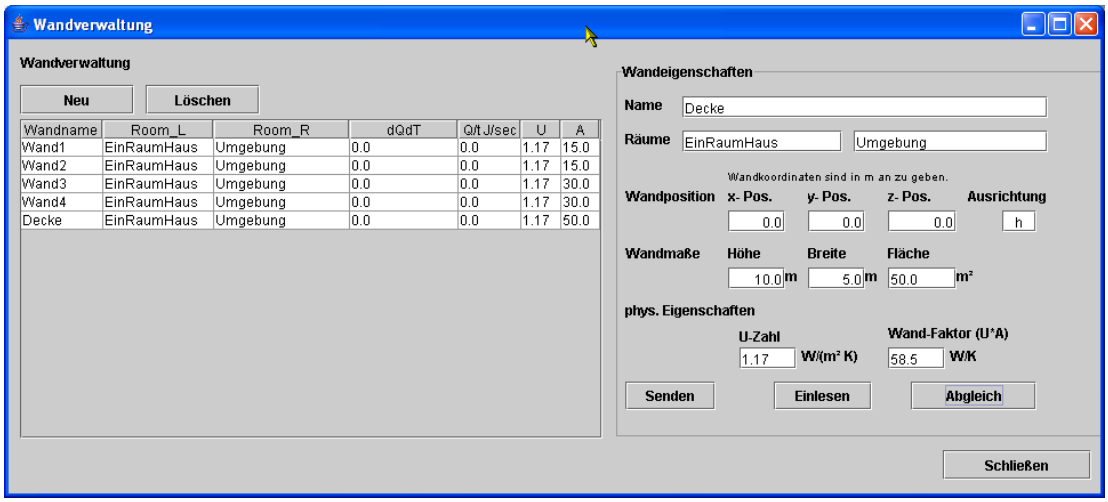


Abb. 42 Gui der Wandeingabe

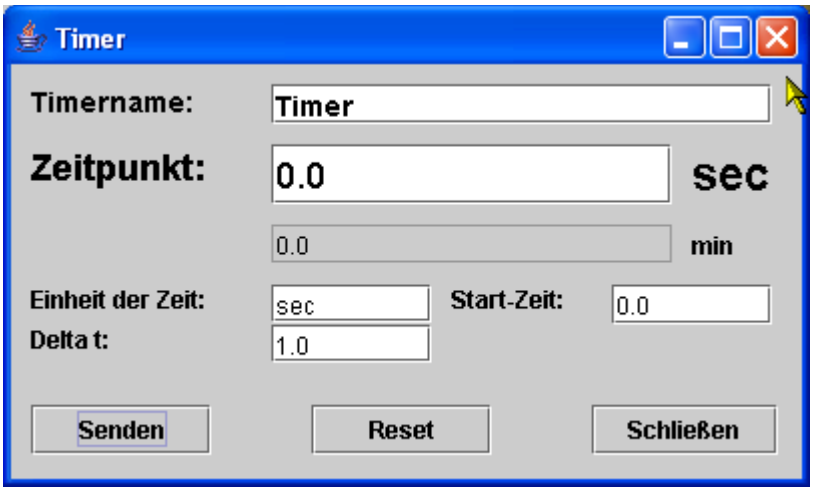


Abb. 43 GUI des Timers

Anschließend werden die Simulationsparameter festgelegt. Die Simulation soll mit einem $\Delta t = 1\,sec$ durchgeführt werden.

Das Messgerät wurde wie folgt konfiguriert:

x-Achse:	0...5000 sec
Einheitenstriche:	alle 100 sec
y-Achse:	-10...30 °C
Einheitenstriche:	alle 5 °C

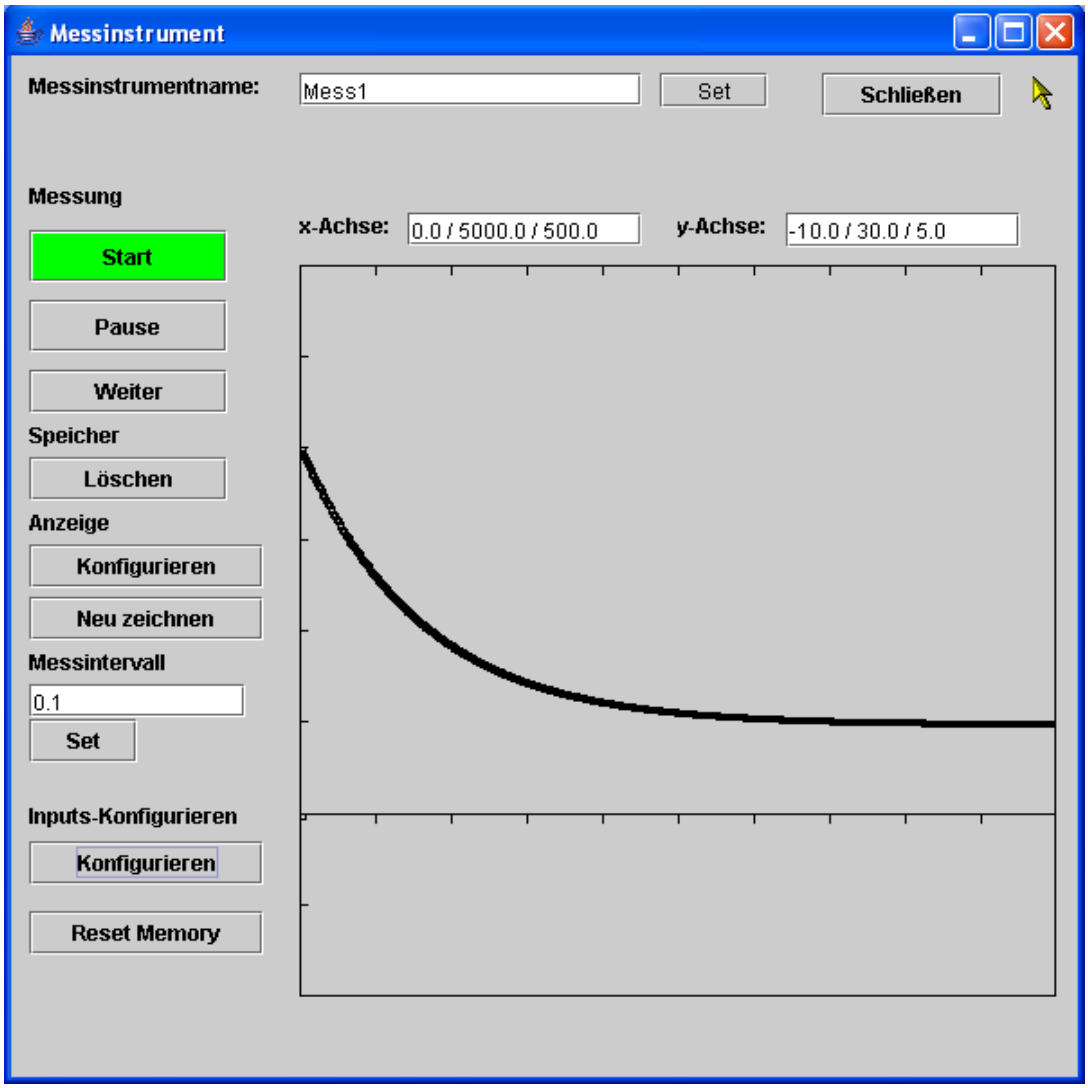


Abb. 44 Zimmertemperaturverlauf im Ein-Raum-Haus

In der Abb. 44 ist der Temperaturverlauf im Ein-Raum-Haus dargestellt.

In einem weiteren Simulationslauf wurde die Konfiguration der Raumobjekte beibehalten. Das Messgerät wurde so konfiguriert, dass der Temperaturverlauf im Raum (schwarz) gleichzeitig mit dem Wärmefluss durch die Wand 1 (rot), Wand 3 (cyan) und die Decke (grün) aufgezeichnet wurde. Es ist zu beachten, dass die dargestellten Werte der Wände und der Decke im Diagramm mit dem Faktor 0,1 gestaucht dargestellt werden.

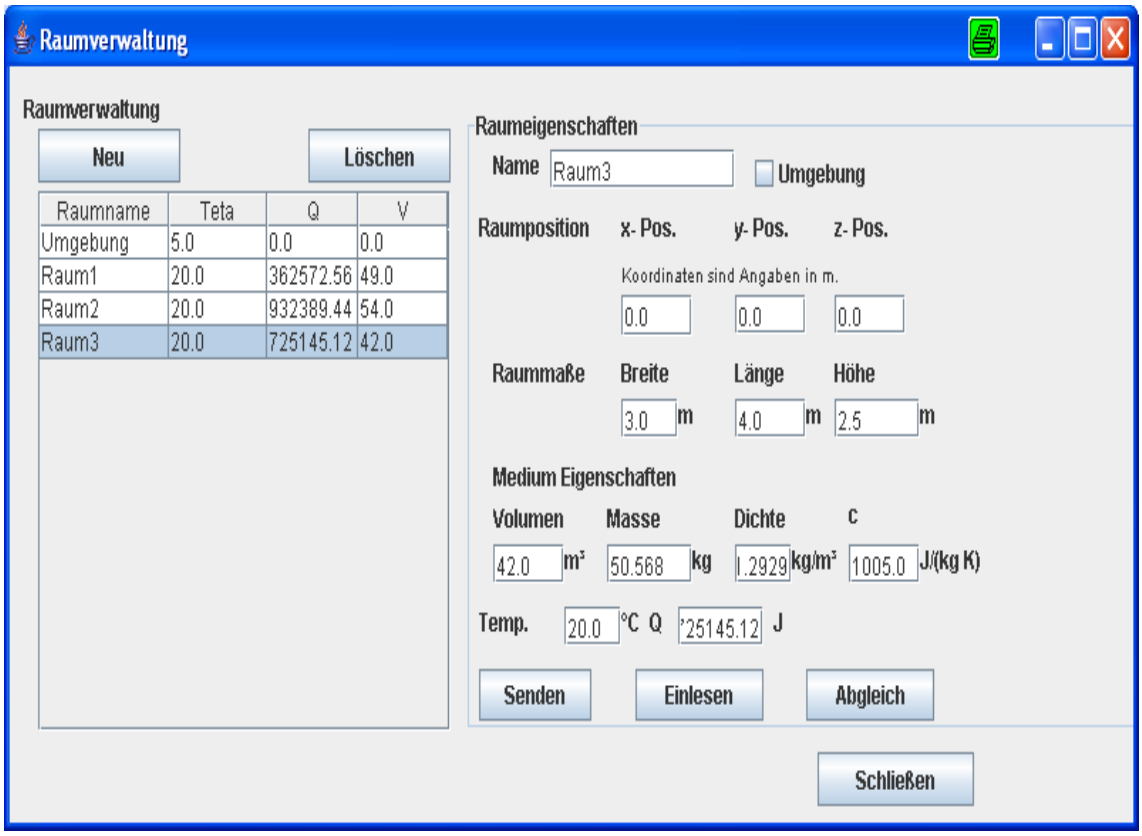


Abb. 45 Wämeverlust durch die Außenwände

schwarz	Temperatur im Raum
rot	Wand 1
cyan	Wand 3
grün	Decke

8.2 Beispiel 2: Bungalow mit 3 Zimmern, Flachdach

Nach dem ersten einfachen Beispiel soll nun anhand eines komplexen Beispiels gezeigt werden, wie man sich einem realen Haus annähern kann. Es handelt sich um einen Bungalow mit 3 Zimmern. Er besteht aus einer Etage und hat ein Flachdach. Die Wärmeverluste durch den Fußboden werden vernachlässigt. Die weiteren Daten können aus den folgenden Tabellen und den Zeichnungen entnommen werden.

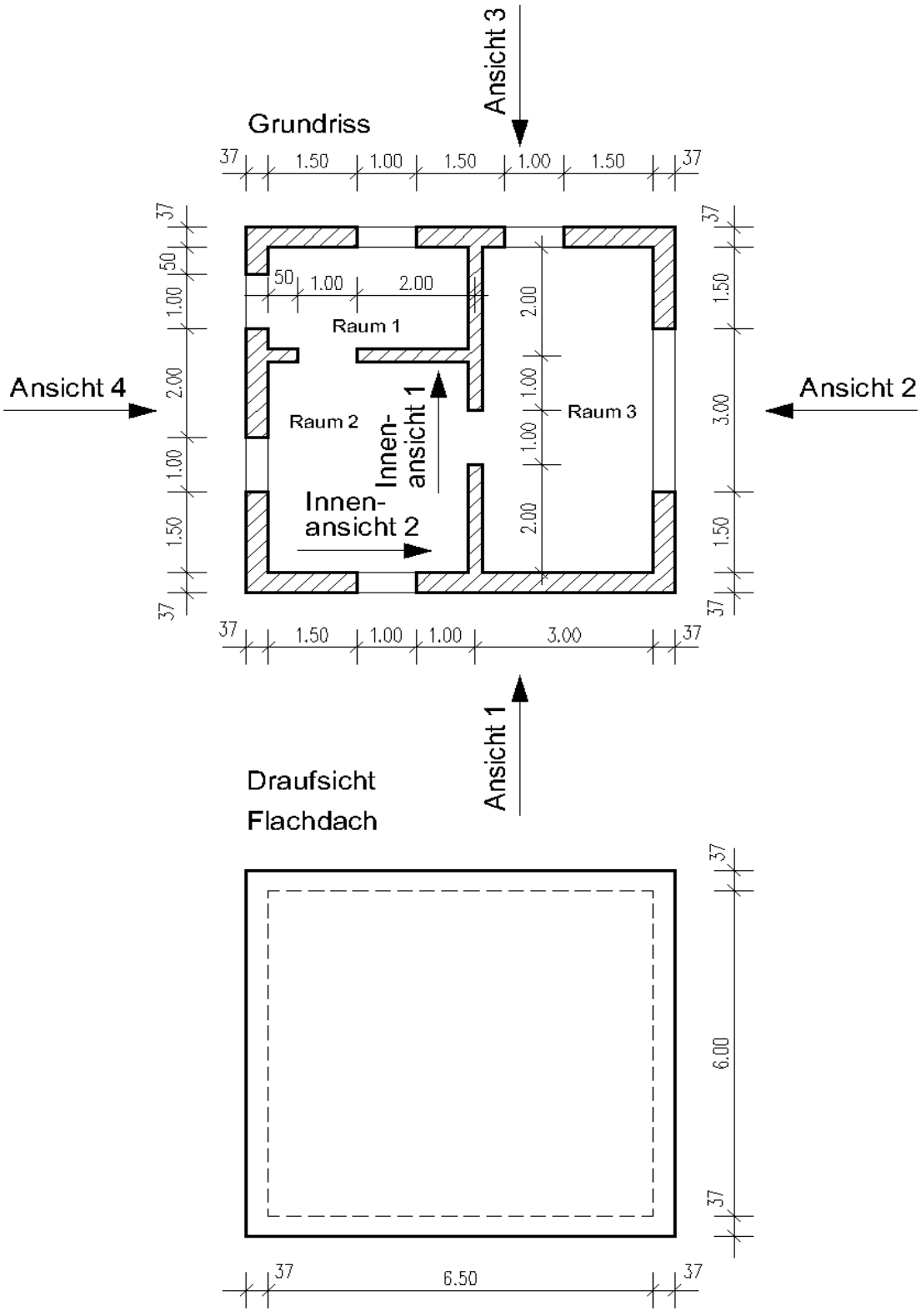


Abb. 46 Grundriss des Bungalows

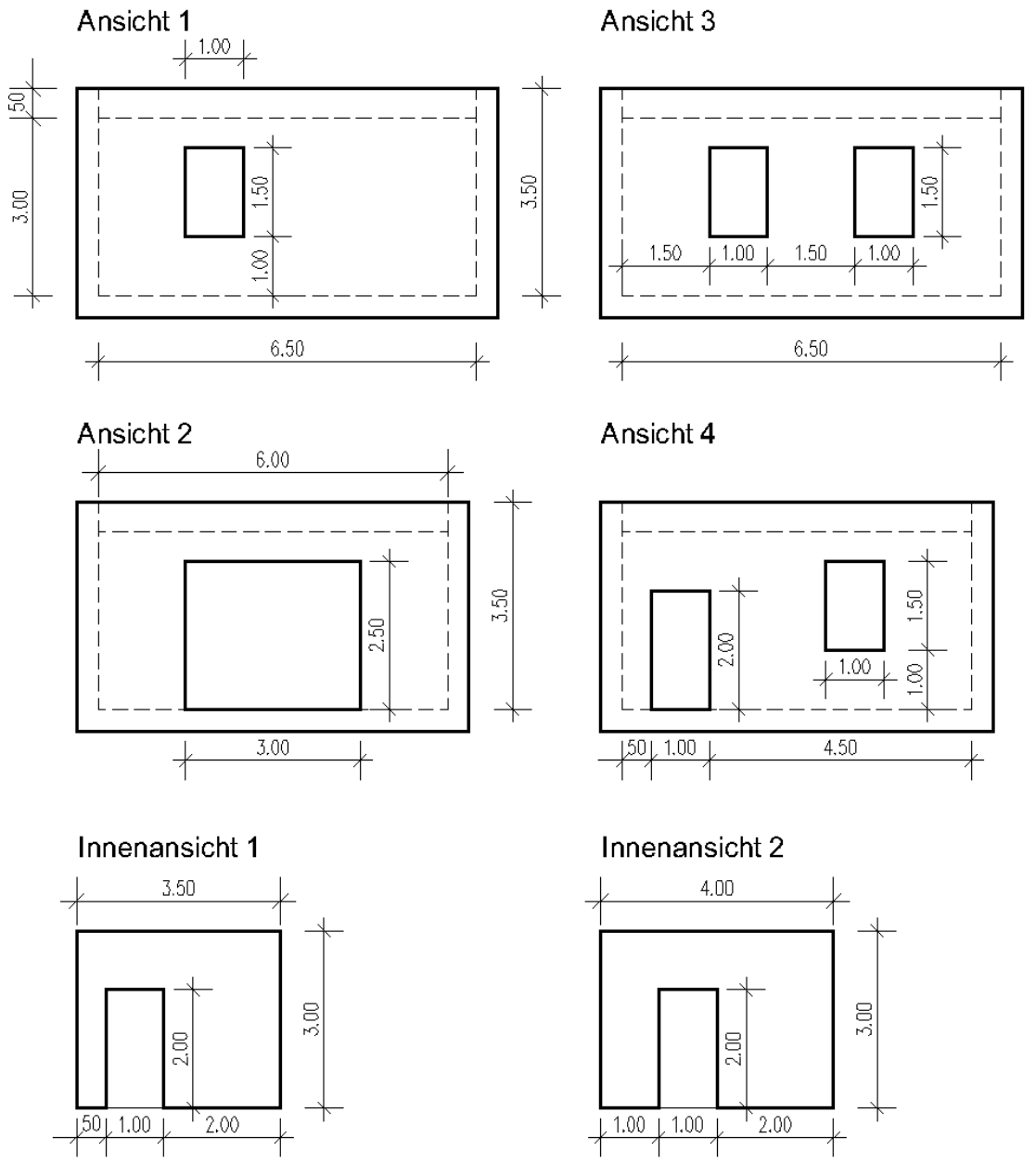


Abb. 47 Außen- und Innenansichten des Bungalows

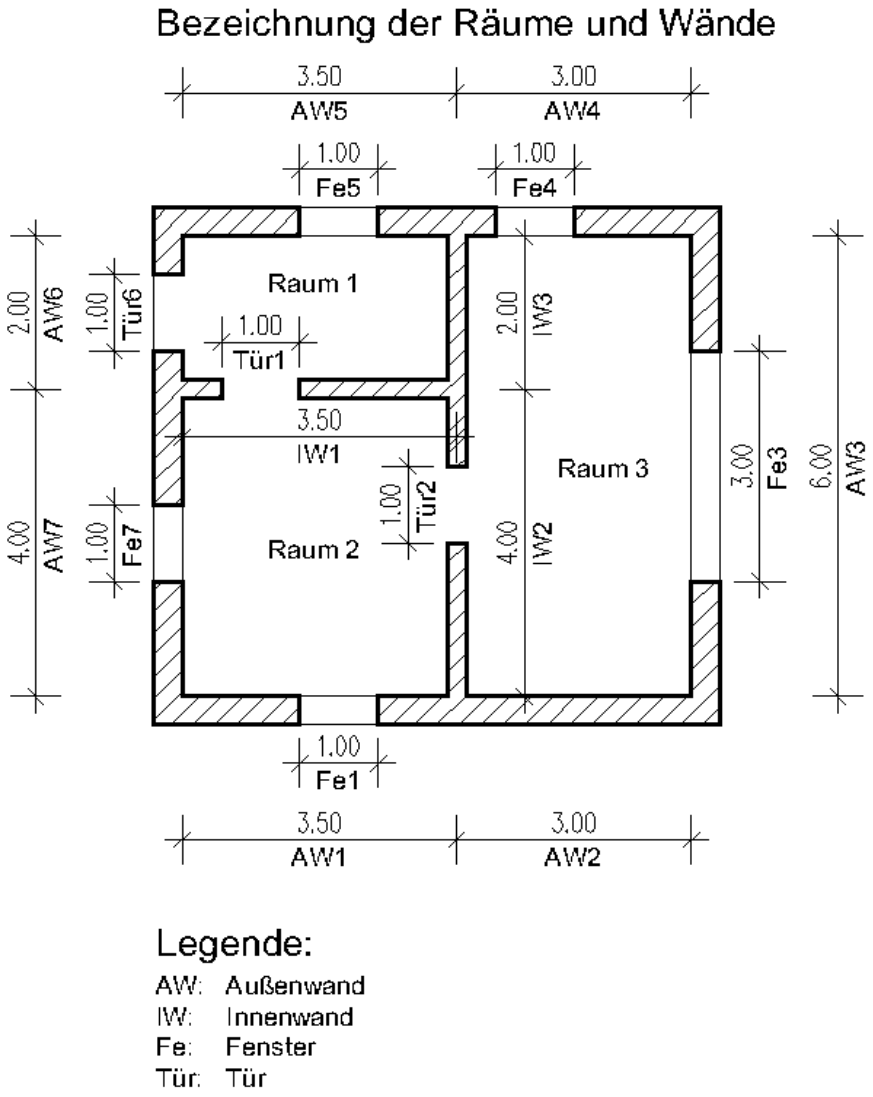


Abb. 48 Benennung der Wände

Raum1:	
Volumen:	$V_{Raum1} = 3{,}5 \cdot 2 \cdot 3\, m^3 = 21\, m^3$
Raum 2:	
Volumen:	$V_{Raum2} = 3{,}5 \cdot 4 \cdot 3\, m^3 = 42\, m^3$
Raum3:	
Volumen:	$V_{Raum3} = 3 \cdot 6 \cdot 3\, m^3 = 54\, m^3$

Dach		
Fläche	$A_{Dach} = 6{,}5 \cdot 6\, m^2 = 39\, m^2$	
Höhe	$h_{Dach} = 0{,}5\, m$	
Wärmedurchgangskoeffizient mit Steinwolle isoliert	$U_{Dach} = 0{,}32 \frac{W}{m^2 \cdot K}$	
Dach1:		Fläche über Raum1
Fläche:	$A_{Dach1} = 3{,}5 \cdot 2\, m^2 = 7\mathrm{m}^2$	
Dach2:		Fläche über Raum 2
Fläche:	$A_{Dach2} = 3{,}5 \cdot 4\, m^2 = 14\mathrm{m}^2$	
Dach3:		Fläche über Raum 3
Fläche:	$A_{Dach3} = 3 \cdot 6\, m^2 = 18\, m^2$	
AW1: Außenwand 1		
Fläche:	$A_{AW1} = (3 \cdot 3{,}5)\, m^2 - (1 \cdot 1{,}5)\, m^2 = 9\, m^2$	Wand-Fenster
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 m	$U_{AW} = 1{,}37 \frac{W}{m^2 \cdot K}$	
Fe1: Fenster		

Fläche	$A_{Fe1}=1\cdot1{,}5\,m^2=1{,}5\,m^2$	
Doppelfenster	$U_{FE}=3{,}3\frac{W}{m^2\cdot K}$	
AW2:		
Fläche	$A_{AW2}=(3\cdot3)m^2$	
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 m	$U_{AW}=1{,}37\frac{W}{m^2\cdot K}$	
AW3: Außenwand 3		
Fläche:	$A_{AW3}=(6\cdot3)m^2-(3\cdot2{,}5)\,m^2=10{,}5\,m^2$	Wand-Fenster
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 cm	$U_{AW}=1{,}37\frac{W}{m^2\cdot K}$	
Fe3: Fenster		
Fläche	$A_{Fe3}=3\cdot2{,}5\,m^2=7{,}5\,m^2$	
Doppelfenster	$U_{FE}=3{,}3\frac{W}{m^2\cdot K}$	
AW4: Außenwand 4		
Fläche:	$A_{AW4}=(3\cdot3)m^2-(1\cdot1{,}5)\,m^2=7{,}5\,m^2$	Wand-Fenster
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 cm	$U_{AW}=1{,}37\frac{W}{m^2\cdot K}$	
Fe4: Fenster		
Fläche	$A_{Fe4}=1\cdot1{,}5\,m^2=1{,}5\,m^2$	
Doppelfenster	$U_{Fe}=3{,}3\frac{W}{m^2\cdot K}$	

AW5: Außenwand 5		
Fläche:	$A_{AW5} = (3{,}5 \cdot 3)\,m^2 - (1 \cdot 1{,}5)\,m^2 = 9\,m^2$	Wand-Fenster
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 cm	$U_{AW} = 1{,}37 \frac{W}{m^2 \cdot K}$	
Fe5: Fenster		
Fläche	$A_{Fe5} = 1 \cdot 1{,}5\,m^2 = 1{,}5\,m^2$	
Doppelfenster	$U_{Fe} = 3{,}3 \frac{W}{m^2 \cdot K}$	
AW6: Außenwand 6		
Fläche:	$A_{AW6} = (2 \cdot 3)\,m^2 - (1 \cdot 2{,}5)\,m^2 = 3{,}5\,m^2$	Wand-Tür
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 cm	$U_{AW} = 1{,}37 \frac{W}{m^2 \cdot K}$	
Tür6: Eingangstür		
Fläche	$A_{Tür1} = 1 \cdot 2\,m^2 = 2\,m^2$	
Holztür außen	$U_{Tür\,außen} = 4{,}1 \frac{W}{m^2 \cdot K}$	
AW7: Außenwand 7		
Fläche:	$A_{AW7} = (4 \cdot 3)\,m^2 - (1 \cdot 1{,}5)\,m^2 = 10{,}5\,m^2$	Wand-Fenster
Wärmedurchgangskoeffizient Kalksand-Lochstein d= 0,37 cm	$U_{AW} = 1{,}37 \frac{W}{m^2 \cdot K}$	
Fe7: Fenster		
Fläche	$A_{Fe7} = 1 \cdot 1{,}5\,m^2 = 1{,}5\,m^2$	

Doppelfenster	$U_{Fe}=3{,}3\frac{W}{m^2\cdot K}$	
IW1: Innenwand 1		
Fläche:	$A_{IW1}=(3{,}5\cdot 3)m^2-(1\cdot 2{,}5)m^2=8\,m^2$	Wand-Tür
Wärmedurchgangskoeffizient Langlochziegel d= 0,24 cm	$U_{IW}=1{,}63\frac{W}{m^2\cdot K}$	
Tür1: Innentür		
Fläche	$A_{Tür2}=1\cdot 2\,m^2=2\mathrm{m}^2$	
Holztür innen	$U_{Tür\,innen}=3{,}8\frac{W}{m^2\cdot K}$	
IW2: Innenwand 2		
Fläche:	$A_{IW2}=(4\cdot 3)m^2-(1\cdot 2{,}5)m^2=9{,}5\,m^2$	Wand-Tür
Wärmedurchgangskoeffizient Langlochziegel d= 0,24 cm	$U_{IW}=1{,}63\frac{W}{m^2\cdot K}$	
Tür2: Innentür		
Fläche	$A_{Tür3}=1\cdot 2\,m^2=2\,m^2$	
Holztür innen	$U_{Tür\,innen}=3{,}8\frac{W}{m^2\cdot K}$	
IW3: Innenwand 3		
Fläche:	$A_{IW3}=(2\cdot 3)m^2=6\,m^2$	
Wärmedurchgangskoeffizient Langlochziegel d= 0,24 cm; innen	$U_{IW}=1{,}63\frac{W}{m^2\cdot K}$	

Die Abb. 49 und Abb. 50 zeigen die GUIs für die Raum- bzw. Wandeingaben.

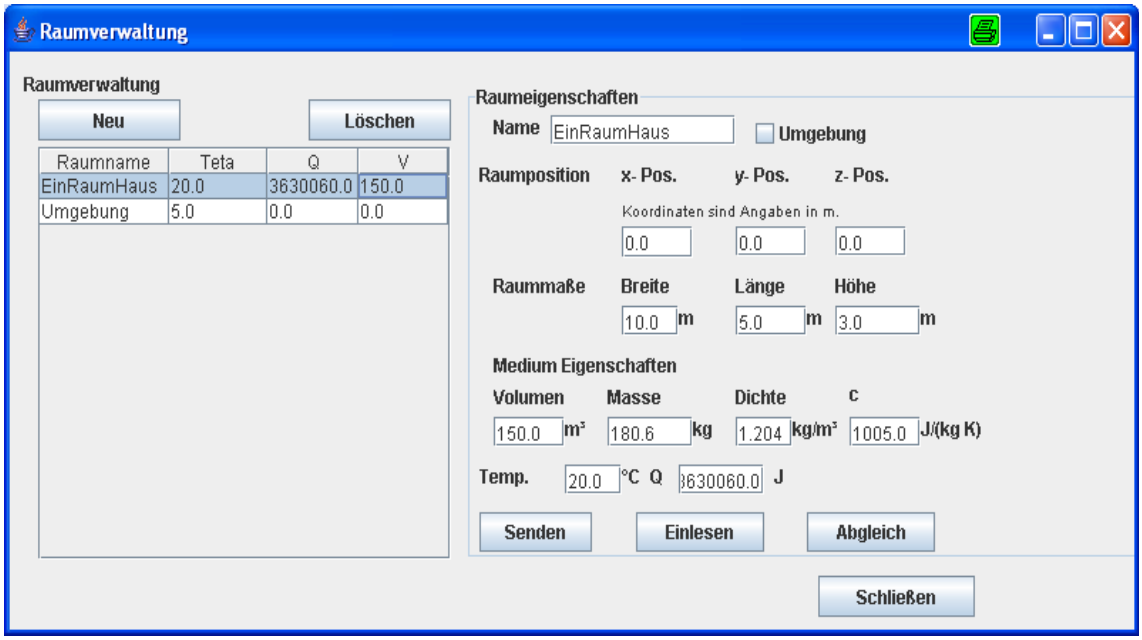


Abb. 49 GUI der Raumeingabe

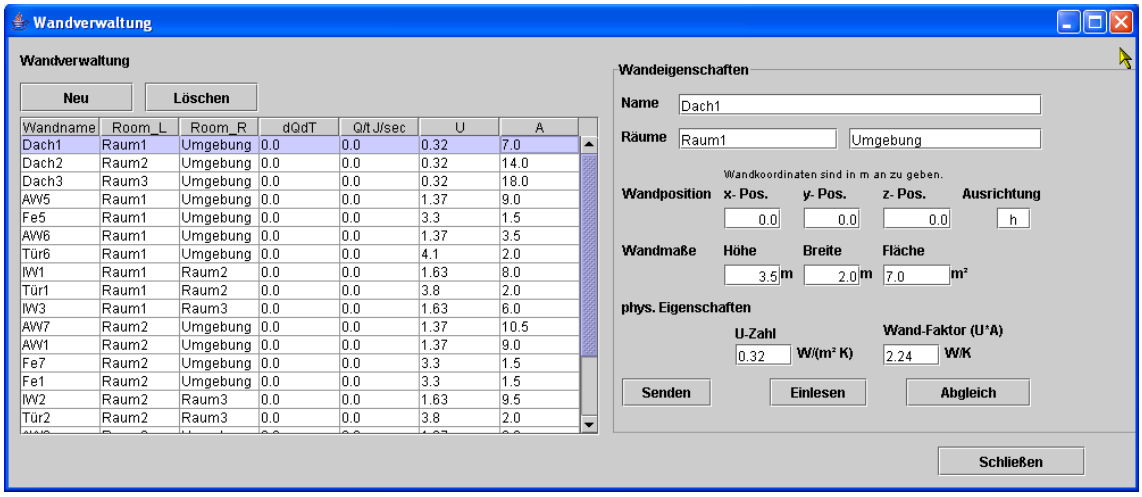


Abb. 50 GUI der Wandeingabe

Die Screenshots Abb. 51, 52 und 53 zeigen beispielhaft die Flexibilität einer kleinen Auswahl von Auswertungsmöglichkeiten.

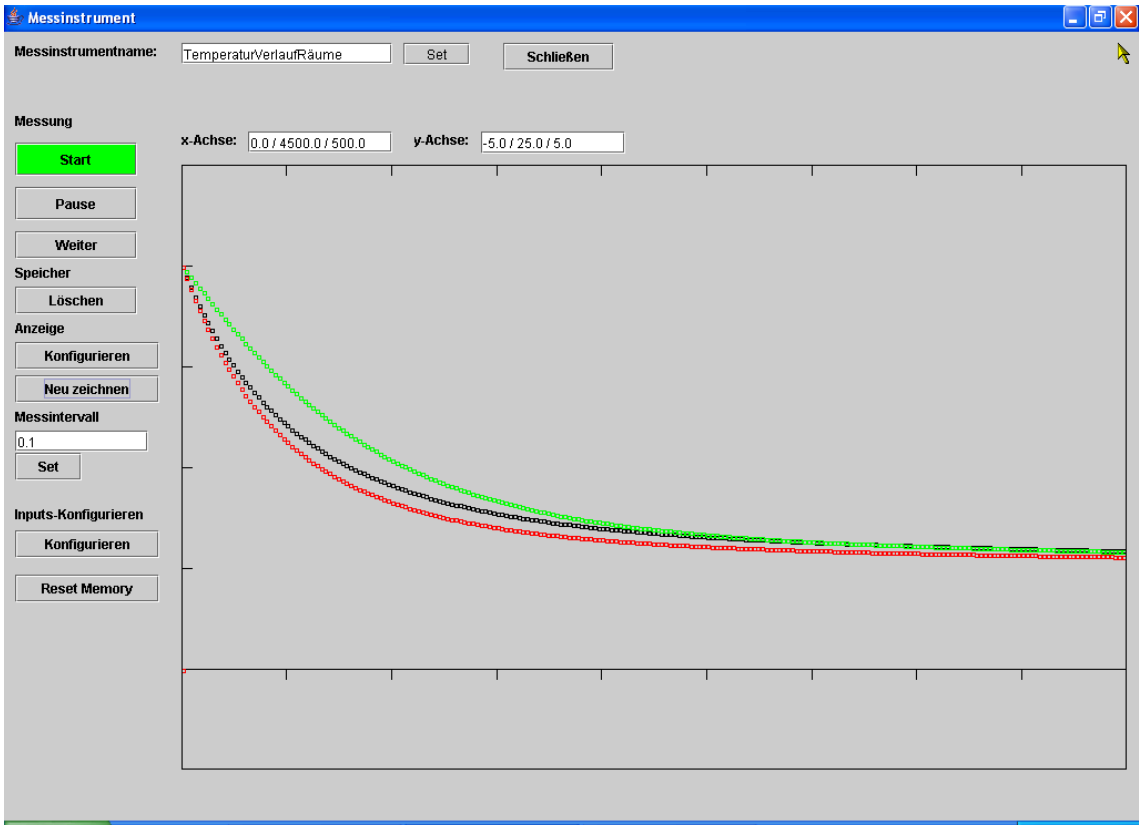


Abb. 51 Temperaturverlauf der Räume

In Abb. 51 ist der Temperaturverlauf der Räume 1 (schwarz), 2 (grün) und 3 (rot) dargestellt.

In Abb.: 52 wird Raum1 betrachtet. Im Graphen ist der Wärmefluss aus Raum1 durch die einzelnen Außenwände aufgezeichnet.

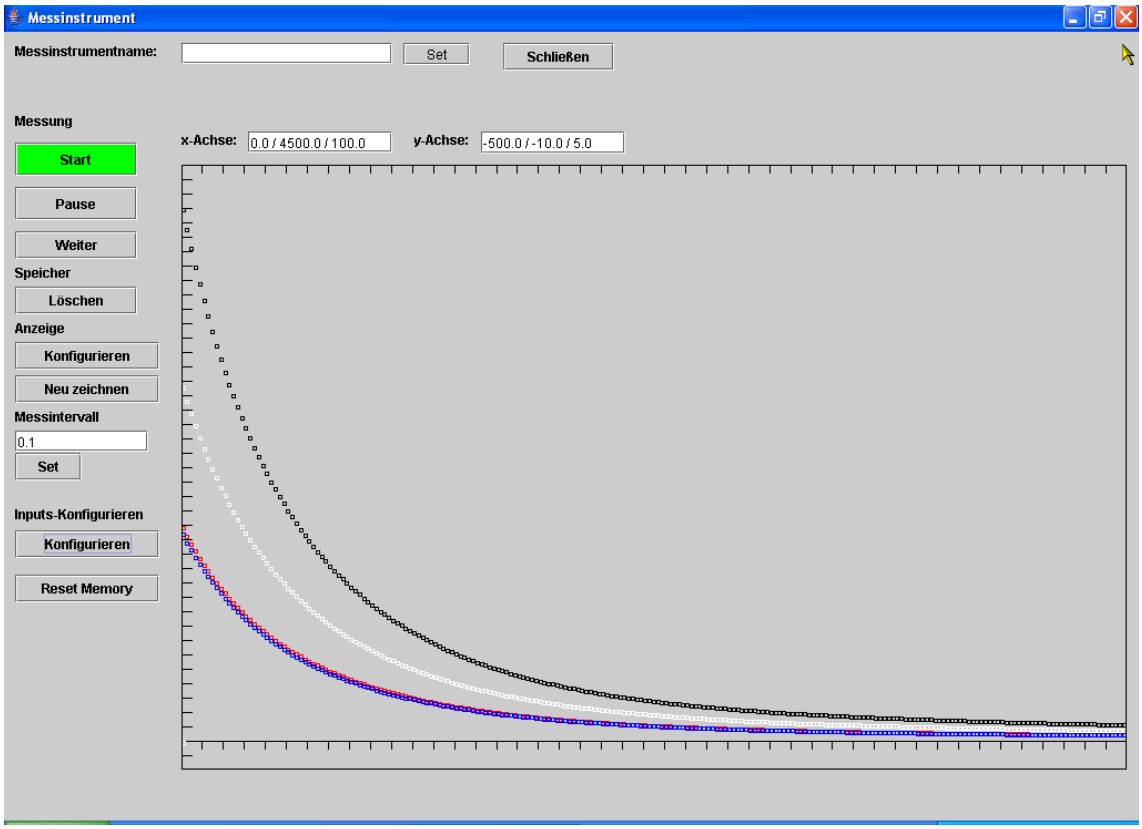


Abb. 52 Wärmeverlust durch die Außenwände des Raumes 1

Die Tabelle enthält die Zuordnung der Farben zu den Wänden.

schwarz	AW5
rot	Fe5
blau	AW6
weiß	Tür6

Im nächsten Graphen werden die Wärmeströme durch die Innenwände betrachtet. Dabei ist auf Grund der Festlegung bei Eingabe der Wände folgende Strömungsrichtung festgelegt worden.

- IW 1 (schwarz) und Tür 1 (grün) von Raum 1 nach Raum 2
- IW 2 (rot) und Tür 2 (blau) von Raum 3 nach Raum 2
- IW 3 (weiß) von Raum 3 nach Raum 1

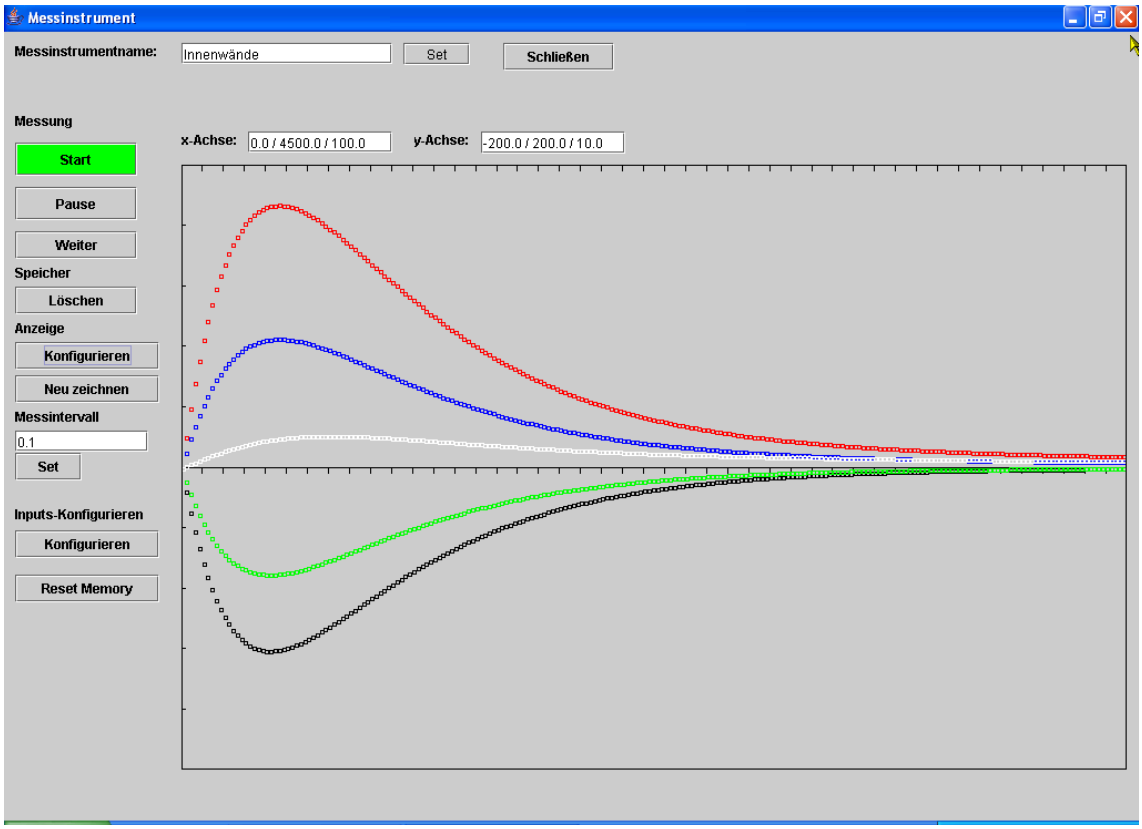


Abb. 53 Wärmefluss durch die Innenwände

IW 1	scharz
Tür 1	grün
IW 2	rot
Tür 2	blau
IW 3	weiß

In dem nachfolgenden Graphen Abb. 54 wird das Heizverhalten eines Heizgerätes in Raum 2 betrachtet. Dabei wurde das Heizgerät im Modus 4, mit 2 Punktregelung (min. 20°C und max. 23 °C) und einer Vorlauftemperatur von 70 °C betrieben. Die Rücklauftemperatur, der obere Regelpunkt (23 °C) und das Verhalten der Raumtemperatur sind aufgezeichnet worden.

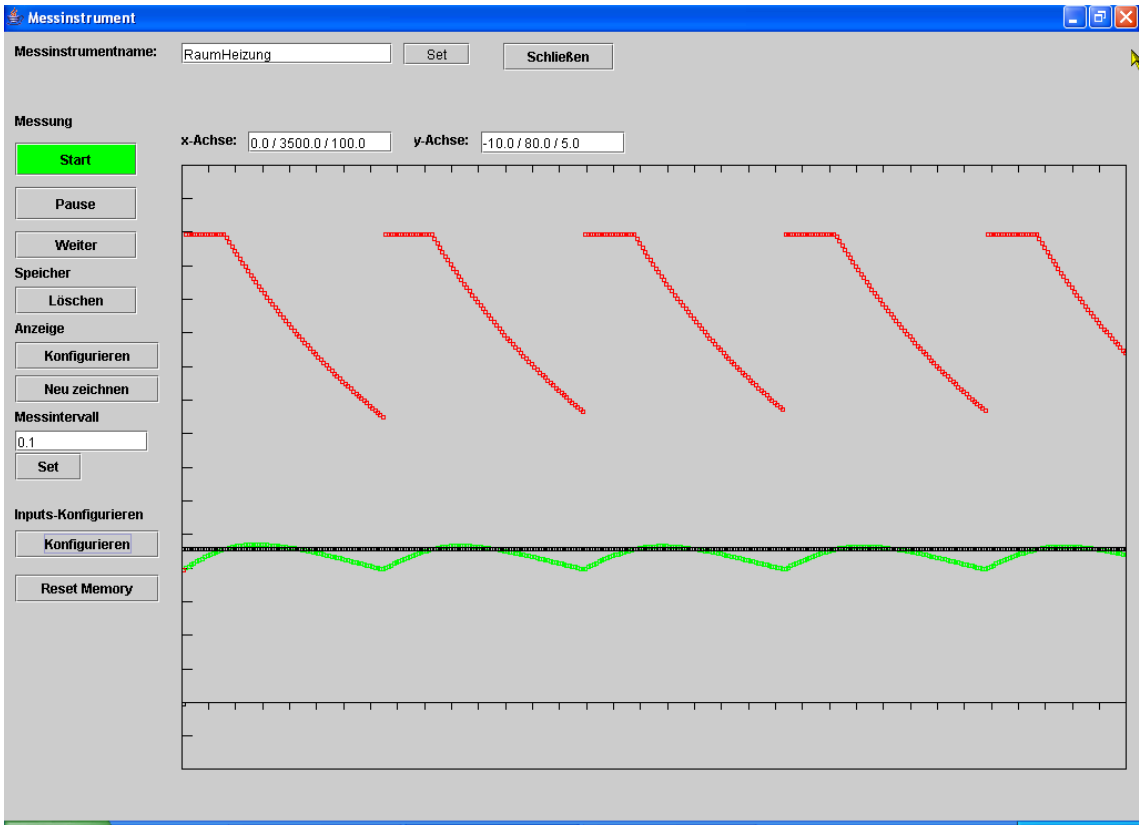


Abb. 54 Regelverhalten der Heizung

rot	Wassertemperatur im Heizkörper
grün	Raumtemperatur
schwarz	oberer Regelpunkt

Diese Beispiele können nur einen kleinen Ausschnitt aus den Auswertungsmöglichkeiten zeigen, die durch die Messgeräte geboten werden. Während der Laufzeit der Simulation können beliebig viele Messgeräte mit unterschiedlichen Auswertungen parallel mitlaufen. Wie im ersten Beispiel gezeigt, können auch Größen mit unterschiedlichen Wertebereichen über unterschiedliche Faktoren so angepasst werden, dass sie in einem Graphen dargestellt werden können.

9 Die IDE *NetBeans*

Eine Frage ist bei der Vorstellung des Konzeptes noch offen: „Wie wird der Programmierer bei der **OOP** durch Tools unterstützt?“

Durch die sehr komplexe Struktur einer OOP-Anwendung scheint es sehr schwierig und bei größeren Projekten fast unmöglich zu sein, den Überblick zu behalten. Es sei denn, es wird per Hand Protokoll geführt. Dieses Problem soll durch sogenannte **IDEs** (Integrated Development Environments) gelöst werden. Diese IDEs unterstützen den Programmierer bei sehr vielen Problemen. Die Objekte werden z. B. sortiert und nach Packages, Klassen, Eigenschaften, Attributen, Konstruktoren, Methoden und Beans gegliedert dargestellt. GUIs können über Formulareditoren graphisch zusammengesetzt werden. Der Quelltext der Objekte wird mit diversen Hilfen dargestellt. Während der Eingabe wird der Quelltext auf Syntaxfehler geprüft, Methoden mit Beschreibung werden zu ausgewählten Objekten aufgelistet und vieles mehr. Dies sind nur ein paar Beispiele, die zeigen wie mächtig moderne IDEs geworden sind. Diese IDEs befinden sich jedoch in einer ständigen Weiterentwicklung, so dass in Zukunft noch einige nützliche Unterstützungsfeatures zu erwarten sind. Die meisten IDEs sind sehr teuer, und können somit aus Kostengründen für den normalen Lehrbetrieb an öffentlichen Bildungseinrichtungen nicht zur Verfügung gestellt werden. Die Lernenden haben oft nicht die finanziellen Mittel um sich eine eigene IDE anzuschaffen. Die Lösung dieses Problems bietet die OpenSource-Community im Internet. Es gibt zwei bemerkenswerte Produkte und zwar **Eclipse**, unter anderem gesponsort von der Firma **IBM**, und **NetBeans**, gesponsort von der Firma **Sun**. Diese Produkte sind im Netz frei downloadbar und unterliegen einer abgewandelten OpenSource-Lizenz. Für den Anwender bedeutet dies, dass er diese Software für den Bildungsbereich frei nutzen darf. Die **SiMo-Umgebung** wurde mit **NetBeans** entwickelt. NetBeans war zu Beginn dieser Arbeit die einzige OpenSource IDE, in der einFormeditor integriert war. Da das Erstellen von Formularen ohne Editor recht zeitaufwändig ist, war die Entscheidung leicht. Dafür bot **Eclipse** andere interessante Features, die NetBeans damals noch nicht

hatte. Beide Produkte werden ständig weiterentwickelt und fast vierteljährlich erscheinen neue Releases, so dass es ratsam ist, sich vor der Entscheidung für eine IDE über die neuen Features beider IDEs zu informieren.

Durch die graphische Darstellung der Objekte und den Hilfen der IDEs wird es selbst dem Programmieranfänger leicht gemacht, sich in bestehende Systeme einzuarbeiten und diese später zu erweitern oder gar selbst neue Projekte zu programmieren. Für das erfolgreiche Einarbeiten in den Quelltext der **SiMo-Umgebung *HausSim*** sind Grundkenntnisse in **OOP** und einer **IDE** erforderlich. Sind diese Grundlagen einmal geschaffen, kann der Lerner mit diesem Wissen sein technisches Verständnis auf andere Problemstellungen übertragen und mit dem hier vorgestellten Konzept seine technischen Handlungskompetenzen erfolgreich ausbauen.

10 Zusammenfassung und Ausblick

Im Rahmen diese Arbeit sollte dargelegt werden, wie angefangen bei der Systemtheorie der Technik nach Ropohl, Systeme begreifbar und erfassbar werden Mit der Unified Modeling Language **UML** wird ein Zwischenglied zur Verfügung gestellt, mit dem die abstrakten Systemzusammenhänge aus den theoretischen Überlegungen strukturiert auf funktionaler Ebene konkretisiert werden können. Und schließlich besteht mit Hilfe der objektorientierten Programmierung **OOP** eine Möglichkeit, die theoretischen Überlegungen in eine Anwendung zu transferieren, die es erlaubt, das konkrete System zu simulieren und das Verhalten am Computer virtuell zu studieren. Dabei kann auf höchst komplexe und zumeist nur eingeschränkt gültige mathematische Formeln und Zusammenhänge verzichtet werden. Die Komplexität wird durch die Rechengeschwindigkeit des Computers kompensiert. Dies bedeutet nicht, dass durch die Darstellung des Systems in der SiMo-Umgebung das System und die Systemzusammenhänge einfacher werden. Der beliebige Ausbau und die Veränderungsmöglichkeiten des Systems ermöglichen sehr komplexe Systeme im Computer abzubilden. Diese Komplexität belastet jedoch nicht unmittelbar den User bei seinem Handeln. Er kann mit einfachen überschaubaren Systemen arbeiten und daraus komplexe Systeme konstruieren. Somit wird dem User ermöglicht, ein System zu konstruieren und während der Simulation an diesem System Verhaltensweisen zu erfahren, die er vielleicht vorher nicht unbedingt erwartet hätte.
Der Lerner erlernt mit diesem Konzept eine strukturierte Vorgehensweise, um der er seine technische Handlungskompetenzen zu erweitern und auch an neue Probleme systematisch und zielsicher heranzugehen.
Aus diesen Gründen wäre es ratsam, dass das Erlernen einer objektorientierten Sprache für technische Fächer zum Standard wird.

Eine zukunftsorientierte technischen Grundbildung sollte in der Schule mit technischem Systemdenken, Problemlöseverfahren und kleinen Programmieraufgaben beginnen, im Laufe der Schullaufbahn erweitert werden, und später im Studium mit anspruchsvolleren Fachinhalten ausgebaut wird. Somit müssten die

Methoden des technischen Denkens und die Umsetzung in die Programmierung nicht erst während des Studiums erarbeitet werden, sondern wären jederzeit abrufbar.

11 Danksagung

An dieser Stelle möchte ich all denjenigen meinen Dank aussprechen, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Professor rer. nat. W. Haupt, der mir die Erstellung dieser Dissertation ermöglichte und jederzeit als Ansprechpartner zur Verfügung stand. Er begleitete die Erstellung dieser Arbeit mit großem Interesse und unterstützte mich mit seiner fachlichen Kompetenz und seinem großen Erfahrungsschatz.

Ich danke Herrn Prof. Dr. -Ing. E. Sauer, dass er sich freundlicherweise als Gutachter zur Verfügung gestellt hat und für seine Unterstützung während der Projektarbeit und bei der Erstellung dieser Dissertation. Des Weiteren für die Ermöglichung der Teilnahme an zahlreichen Messen, Tagungen und Seminaren, wodurch ich zahlreiche Denkanstöße und weitere Anregungen für diese Arbeit erhielt.

Weiterhin gilt mein Dank den wissenschaftlichen Mitarbeitern und Projektkollegen Herrn T. Langkau, M. A. und Herrn Dr. -Ing. J. Wehling für die zahlreichen fruchtbaren Diskussionen und ihrer fachlichen Unterstützung.

Mein weiterer Dank gilt der Mitarbeiterin im Labor Frau B. Neuhaus, den Mitarbeittern in den Werkstätten Herrn M. Ferber und Herrn C. Gewaltig für ihre große Unterstützung in den praxisbezogenen Fragestellungen.

Nicht zu vergessen gilt mein Dank auch unseren Projektpartnern vom ATP (Allgemeine Technik Pädagogik, TU-Braunschweig) unter der Leitung von Prof. Dr. -Ing. W. Theuerkauf.

Des Weiteren möchte ich mich bei den Institutionen der Universität Duisburg-Essen, Campus Essen für die gute Zusammenarbeit bedanken. Insbesondere bei den Mitarbeitern des MZ (Medienzentrum) und des HRZ (Hochschulrechenzentrum).

Abschließend gilt mein Dank allen weiteren Gesprächspartnern innerhalb der Universität, auf den Messen, Tagungen und den Seminaren, die mir durch ihre offene, konstruktive Kritik und ihre Anregungen sehr geholfen haben.

Besonderer Dank gilt meinem Freund Herrn Dipl. -Wirt. Inform. B. L. Dewanto (Uni- Münster), der mir sehr hilfreiche Hinweise bezüglich der Java-Programmierung geben konnte.

12 Abkürzungen

ATP	Allgemeine Technik Pädagogik (TU-Braunschweig)
BMBF	Bundesministerium für Bildung und Forschung
GUI	General User Interface
IDE	Integrated Development Environment
Java	eine Programmiersprache der Firma **Sun microsystems, Inc**
MMDB-TU	MultiMediaDatenbank-Technikunterricht
OOP	objektorientierte Programmierung
SiMo-Umgebung	Simulations- u. Modellierungs-Umgebung
TUD	Technologie und Didaktik der Technik (Universität Duisburg-Essen, Campus Essen)
UML	Unified Modeling Language
LiB	Lernobjekte im Baukastenmodus (Projekttitel)

Verwendete Formelzeichen

A	Fläche
h	Höhe
V	Volumen
m	Masse
ρ	Dichte
t	Zeit
ϑ	Temperatur
Q	Wärmemenge
Φ	Wärmefluss
c	
c	Wärmekapazität
U	Wärmedurchgangskoeffizient
α	Wärmeübergangskoeffizient

Anmerkung: Nach der neuen **DIN EN 12831 -** ***Verfahren zur Berechnung der Norm-Heizlast*** wurde das Formelzeichen für den Wärmedurchgangs-Koeffizienten **k-Zahl** in **U** umbenannt. In dieser Arbeit wird die neue Bezeichnung verwendet.

13 Abbildungsverzeichnis

Abbildungen

14 Literaturverzeichnis

[Arp, 1991] Arp, H.: ***Grundkategorien technologischer Beschreibungen, in tu - Technik im Unterricht***, 15. Jg., Heft 59, 1991

[Arp, 1994] Arp, H.: ***Vorlesungsskript zur allgemeinen Technologie (Didaktik der Technik IV)***. Gerhard-Mercator-Universität Duisburg, 1994

[Arp, 2000] Arp, H.: ***Generalisierung technischen Denkens und Handeln – Ansätze aus der Technikgeschichte und der Systemtheorie***, in [Bader, Jenewein, 2000]

[Asano, Bhattachary, Haupt, Wheling, 2001] Asano, R./ Bhattacharya, D./ Haupt W./ Wehling J.: ***Multimediamodul Lichtwellenleiter***, Homepage: http://it.tud.uni-essen.de, 2001

[Bader, Jenewein, 2000] Bader, R./ Jenewein, K. (Hrsg.): ***Didaktik der Technik zwischen Generalisierung und Spezialisierung***, Verlag der Gesellschaft zur Förderung arbeitsorientierter Forschung und Bildung, Frankfurt am Main, 2000

[Balzert, Heide, 2001] Balzert, Heide.: ***Lehrbuch der Objektmodelierung***, Spektrum Akademischer Verlag, Heidelberg-Berlin, 2001

[Balzert, Helmut, 2001(1)] Balzert, Helmut.: ***Lehrbuch der Softwaretechnik – Softwareentwicklung*** (2. Auflage), Spektrum Akademischer Verlag, Heidelberg-Berlin, 2001

[Balzert, Helmut., 2001(2)] Balzert, Helmut.: ***Lehrbuch Grundlagen der Informatik***, Spektrum Akademischer Verlag, Heidelberg-Berlin, 2001

[Bertalanffy, 1949] Bertalanffy, L.: ***Zu einer allgemeinen Systemlehre***, 1949, in [Bleicher 1972]

[BMBF] Homepage des ***Bundesministeriums für Bildung und Forschung***: http://www.bmbf.de

[Bresges, 2002] Bresges A.: ***Objektorientierte Modellbildung in der naturwissenschaftlichen und technischen Bildung***, Dissertation in der Fakultät für Naturwissenschaften der Gerhard-Mercator-Universität Duisburg, 2002

[Brugger, 2001] Brugger, W.: ***Die Erstellung von wiederverwendbaren Inhalten für Web-basierte Kurse***, Virtueller Campus: Szenarien – Strategien – Studium, Wagner, E, Kindt M. (Hrsg.) Medien in der Wissenschaft, Band 14, Waxmann Verlag Münster, 2001

[COREJAVA-1, 2002] Horstmann, C. S./ Cornell G.: ***Core Java Band 1 - Grundlagen***, Markt und Technik, München 2002

[COREJAVA-2, 2002] Horstmann, C. S./ Cornell G.: ***Core Java Band 2 - Expertenwissen***, Markt und Technik, München 2002

[Corliss, 2002] Corliss, G.: ***Automatic differentiation of algorithmus***, Springer, New York, 2002

[Dietz, 1977] Dietz, L.: ***Heizlastberechnung***, Verlag für Bauwesen, Berlin, 2000

[DYNASYS] Homepage der Firma www.Hupfeld-Software.de des Produkts ***DYNASYS***: http://www.hupfeld-software.de

[Eckert, 2000] Eckert, M.: ***Theorie technischer Systeme – Ein Ansatz zur Didaktik technisch-beruflicher Fachrichtungen?*** in [Bader, Jenewein, 2000]

[Eclipse] Offizielle Homepage des Projektes ***ECLIPSE***: http://www.eclipse.org

[Erler, 2000] Erler, T.: ***UML – Der methodische und ausführliche Einstieg***, verlag moderne industrie Buch AG & Co. KG, Landsberg, 2000-2001

[Gerber, 1979] Gerber, E.: ***Thermoclimat: Rechenprogramm zur Bestimmung von Wärmebedarf und Kühllast***, Müller-Verlag Karlsruhe, 1979

[Hansen, 2002] Hansen, T.: ***Inhaltliche und formale Anforderungen: Erfahrungen bei der Content-Erstellung zu einem dynamischen Lernsystem***, in [Lit 2002]

[Harreis, 2000] Harreis, H.: ***Strukturorientiertes Denken und Modelle in den Naturwissenschaften und in der Technik***, in [Bader, Jenewein, 2000]

[Haupt, Sanfleber] Haupt, W/ Sanfleber, H.: ***Ansatz zu einer Didaktik der Technik für den Technikunterricht in der Sekundarstufe II (Differenzierte Gymnasiale Oberstufe)***, in Traebert ; Spiegel (Hrsg.): Technik als Schulfach, Düsseldorf 1979

[Heider, 1991] Heider-Hasebrink, Frithilde: ***Explizites versus implizites Wissen und Lernen, Dissertation im Fachbereich Pädagogik der Universität der Bundeswehr Hamburg***, 1991

[Hillen, Paul, Puschhof, 2002] Hillen, S./ Paul, G./ Puschhof, F.: ***Systemdynamische Lernumgebungen – Modellbildung und Simulation im kaufmännischen Unterricht***, Frankfurt a. Main, 2002

[ILIAS] Projekthomepage der ***Lernplattform ILIAS*** http://www.ilias.uni-koeln.de

[Issing, 1997] Issing, L. J./ Klimsa P. (Hrsg.): ***Information und Lernen mit Multimedia***, (2.Auflage), Psychologie Verlags Union, Weinheim, 1997

[Issing, 1998] Issing L. J.: ***Lernen mit Multimedia aus psychologisch-didaktischer Perspektive***, in Dörr, G./ Jüngst, Karl (Hrsg.): Lernen mit Medien. Ergebnisse und Perspektiven zu medial vermittelten Lehr- und Lernprozessen. Juventa, Weinheim, 1998

[Krause, 2002] Krause, D.: ***Aspekte der Mediennutzung im offenen Seminarkonzept***, in Wolf-Gideon Bleek u. a. (Hrgs.), Medienunterstütztes Lernen, 2002, Homepage: http://www.wisspro.de

[Kuchling1994] Kuchling, H.: ***Taschenbuch der Physik, Fachbuchverlag Leipzig-Köln***, 1994

[Langkau, Rudolph, Wehling, 2003] Langkau, T./ Rudolph, C./ Wehling J.: ***Selbstreguliertes Lernen mit neuen Medien***, in [LIT2003]

[Langkau, Rudolph, Wehling, Haupt, 2002] Langkau, T./ Rudolph C./ Wehling J./ Haupt W.: ***Multimediales Lernen im Fach Technologie und Didaktik der Technik***, in [Lit 2002]

[Lit, 2002] Jantke, K. P./ Wittig, W. S./ Herrmann, J. (Hrsg.): Tagungsband der LIT 2002: ***Von e-Learning bis e-Payment, Das Internet als sicherer Marktplatz***, Akademische Verlagsgesellschaft, AKA , Berlin, 2002

[Lit, 2003] Jantke, K. P./ Wittig, W. S./ Herrmann, J. (Hrsg.): Tagungsband der LIT 2003: ***Von e-Learning bis e-Payment 2003, Das Internet als sicherer Marktplatz***, Akademische Verlagsgesellschaft, AKA , Berlin, 2003

[Mandel, 2001] Reinmann-Rothmeier G./ Mandel H.: ***Unterrichten in Lernumgebungen***; in Krapp, A, und Weidmann B. (Hrsg): Pädagogische Psychologie. Ein Lehrbuch (4. Aufl.): Beltz, Weinheim. 2001

[Meier, C., 2003] Meier, C., Mitarbeiter am Fraunhofer-Institut Arbeitswirtschaft und Organisation, Stuttgart/ Seufert, S., Mitarbeiterin am mcm-Institut, Universiät St. Gallen: ***Lebenslanges (E-)Learning: Lust oder Frust?- Zum Potenzial digitaler Lernspiele für die betriebliche Bildung***, erschienen in: Weiterlernen neu gedacht. QUEM-Report, Heft 78, Berlin, 2003

[Meyer, H., 2000(1)] Meyer, H.: ***UnterrichtsMethoden I:Theorieband***, Cornelsen, Frankfurt a. Main, 2000

[Meyer, H., 2000(2)] Meyer, H.: ***UnterrichtsMethoden II:Praxisband***, Cornelsen, Frankfurt a. Main, 2000

[mhSoft] Hompage der Firma ***mh-software*** (Wärmebedarfsberechnungen): http://www.mh-software.de

[Möller, 1992] Möller, D.: ***Modellbildung, Simulation und Identifikation dynamischer Systeme***, Springer, Berlin, 1992

[Montada, 1998] Montada, O.: ***Entwicklungspsychologie***, (4. Auflage), Psychologie Verlags Union, Weinheim, 1998

[NetBeans] Boudreau, T./ Glick, J./ Greene S./Spurlin V./ Woehr J.: ***NetBeans – The Definitive Guide***, O' Reilly & Associates, Inc. Printed in the United States of America, 2003

[NetBeansPrj] Offizielle Homepage des Projektes ***NetBeanas:*** http://www.netbeans.org

[Oestereich, 2001] Oestereich B.: ***Objektorientierte Softwareentwicklung***, 5 ed. Verlag R. Oldenbourg, München, 2001

[Pawlowski, 2001] Pawlowski, J. M.: ***Das Essener-Lern-Modell (ELM), Ein Vorgehensmodell zur Entwicklung computergestützter Lernumgebungen*** Diss. Universität Essen, 2001

[Pfretzschner, Hoppe, 2002] Pfretzschner, R./ Hoppe Thomas: ***Netzbasiertes Lernen und Arbeiten in virtuellen Gemeinschaften***, in [Lit 2002]

[Richtl-1] Ministerium für Schule und Weiterbildung, Wissenschaft und Forschung des Landes Nordrhein-Westfalen (Hrsg.): ***Richtlinien und Lehrpläne für die Sekundarstufe II – Gymnasium/Gesamtschule in Nordrhein-Westfalen. Fach Technik***.: Ritterbach Verlag GmbH, Fechen, 1999

[Ropohl, 1981] Ropohl, G.: ***Interdiziplinäre Technikforschung***, Berlin 1981

[Ropohl, 1999] Ropohl, G.: ***Allgemeine Technologie. Eine Systemtheorie der Technik***, 2. Auflage, Hanser-Verlag, München, 1999

[Sauerbier, 1999] Sauerbier, T.: ***Theorie und Praxis von Simulationssystemen***, Vieweg,Braunschweig, 1999

[Scherf, 2003] Scherf, H. E.: ***Modellbildung und Simulation dynamischer Systeme***, Oldenbourg, München, 2003

[Schmitt, 1999] Schmitt, T.: ***Dynamische Simulation als Werkzeug zur optimierten Planung und Betriebsführung von Abwasserreinigungsanlagen***, Fachgebiet Siedlungswasserwirtschaft, Universität Kaiserslautern. Hrsg. Von T. G. Schmitt, Kaiserslautern, 1999

[Schüler, 2000] Schüler, U.: ***Erkennen von Systemzusammenhängen – Neue Aufgaben und Entwicklungen in der Maschinentechnik***, in [Bader, Jenewein, 2000]

[Seibold, 2002] Seibold, W. ***Ein Mehrprozessorsystem mit rekonfigurierbarer Verbindungsstruktur für die parallele und verteilte ereignisgesteuerte Simulation***, Inst. Für Nachrichtenvermittlung und Datenverarbeitung , Stuttgart, 2002

[SMILE] Homepage der Firma dezentral, Produkt ***SMILE – Eine objektorientierte Software***, die es erlaubt Simulationsmodelle zu erstellen: http://www.dezentral.de

[STELLA] Homepage der Firma iseesystems mit dem Produkt ***Stella***: http://www.iseesystems.comhttp:

[SUN-1] Offizielle Hompage der Firma ***SUN microsystem***: http://www.sun.com

[SUN-2] Offizielle Hompage von ***Java***, der Firma SUN microsystem: http://www.java.sun.com

[Theuerkauf, Haupt, Wagener, Weiner, 1995] Theuerkauf, W. F./ Haupt W./ Wagener W./ Weiner, A.: ***Technology Education in Germany's Gymnasia***, in Langer K./ Metzing, M/ Wahl, D., Technology Education, Innovation and Management, Springer, Heidelberg, 1995

[Traebert, 2000] Traebert, W. E.: ***Technik als Schulfach in der Sekundarstufe I***, in [Bader, Jenewein, 2000]

[TUD-Eingangsportal-1] ***Homepage des Faches TUD*** an der Universität Duisburg-Essen, Campus Essen: http://www.tud.uni-essen.de

[TUD-Eingangsportal-2] ***Projekthomepage LIB des Faches TUD*** an der Universität Duisburg-Essen, Campus Essen: http://lpf.tud.uni-essen.de

[TUD-Eingangsportal-3] Eingansportal zu ***Lernplattform ILIAS von dem Fach TUD*** an der Universität Duisburg-Essen, Campus Essen: http://www.tud.uni-essen.de

[TUD-Eingangsportal-4] Eingansportal zu ***Multimedia Datenbank Technikunterricht MMDB-TU von dem Fach TUD*** an der Universität Duisburg-Essen, Campus Essen: http://www.mmdb-tu.de

[VDI-Richtlinie 3633] Hrsg. VDI-Gesellschaft Fördertechnik Materialfluss Logistik: VDI 3633 Blatt 11; **Simulation von Logistik-, Materialfluss- und Produktionssystemen – Simulation und Visualisierung**; in VDI-Handbuch Materialfluss und Fördertechnik – Band 8; Ausgabedatum 2003-08

[Wagener, Haupt, 2000] Wagener W./ Haupt, W.: ***Technik als Fach der gymnasialen Oberstufe***, in [Bader, Jenewein, 2000]

[Wällnitz, Ficert, Kalfa, 2002] Wällnitz, E./ Fickert, V/ Kalfa W.: ***Möglichkeiten von Simulation und virtuellen Praktika in der Präsenzlehre des Lehrgebietes Betriebssysteme durch e-Learning***, in [LIT 2002]

[Wehling, 2002] Wehling, J.: ***Technische Bildung. Ansätze und Perspektiven: Lehrerausbildung für das Fach Technik im Gymnasialen Bereich Informationstechnologie***, 1. Auflage, Europäischer Verlag der Wissenschaften; 2002

[Wehling, 2003] Wehling, J: ***Eine datenbankbasierte Entwicklungsumgebung zur Online-Generierung von multimedialen Lernmodulen für des Lernen im Fach Technik,*** Dissertation im Fach Technologie und Didaktik der Technik an der Universität Duisburg-Essen, Campus Essen, 2003

[Wehling, Langkau, Rudolph, Haupt, 2003] Wehling, J./ Langkau, T./ Rudolph, C./ Haupt W.: Didaktische Aspekte der Standardisierung und ***Implementierung strukturierter multimedialer Elemente für die Unterstützung eines e-Learning im Fach Technologie und Didaktik der Technik***, in [LIT2003]

[Wentzlaff, 1983] Wentzlaff, G.: ***Weiterentwicklung der Berechnungsverfahren für den Norm-Wärmebedarf und den Jahreswärmeverbrauch von Heizanlagen***, VDI-Verlag, Düsseldorf, 1983

[Wisser, 2002] Wisser B.: ***Computeranimation als Didaktisches Werkzeug – Vorteile, Möglichkeiten, Kosten***, in [Lit 2002]

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema
„Transformation eines technischen Systemmodells auf der Basis der Systemtheorie der Technik nach Ropohl in eine Simulations- und Modellierungs-Umgebung“
selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Willich, den 30. 5. 2005